IRIS® HIPPI Administrator's Guide

Document Number 007-2229-006

CONTRIBUTORS

Written by Carlin Otto and Bill Tuthill
Illustrated by Carlin Otto and Dan Young
Production by Carlos Miqueo
Engineering contributions by Irene Kuffel, Jim Pinkerton, Jean-Michel Pittet,
Jim Peck, Ken Powell, Thomas Skibo, and Brad Strand
St Peter's Basilica image courtesy of ENEL SpA and InfoByte SpA. Disk Thrower
image courtesy of Xavier Berenguer, Animatica.

IRIS® HIPPI Administrator's Guide
Document Number 007-2229-006

Contents

List of Figures

List of Tables

Introduction

This document describes IRIS HIPPI 3.3 for IRIX 6.5 (or later).

The IRIS HIPPI product is a network interface controller board (hardware) and driver and utilities (software) providing ANSI standards-compliant data communication through the High-Performance Parallel Interface (HIPPI). The product provides the following implementations:

- copper-based HIPPI-800 (HIPPI-PH) connectivity for Silicon Graphics systems that have HIO slots (for example, CHALLENGE L and XL, PowerChallenge, Onyx, and Power Onyx platforms), and
- fiber-optic based HIPPI-Serial connectivity for Silicon Graphics systems that have XIO slots (for example, the Origin series and Onyx2 platforms).

The IRIS HIPPI hardware must be installed by a Silicon Graphics system support engineer (SSE) or other person trained by Silicon Graphics. The 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. For Challenge and Onyx systems, use the *IRIS HIPPI Board Installation Instructions*; for Origin and Onyx2 systems, use *IRIS HIPPI-Serial XIO Board Installation Instructions*.

Installation and configuration of the software on all platforms can be done by customers and/or SSEs. This document, *IRIS HIPPI Administrator's Guide* (shipped with the IRIS HIPPI software), provides software configuration details. The online *IRIS HIPPI Release Notes* provide software installation instructions and late-occurring information about the product.

Support for Upper Layer Applications

IRIS HIPPI supports the following upper layer applications:

- standard IRIX 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. See the online *IRIS HIPPI API Programmer's Guide* (shipped with the IRIS HIPPI software) 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.

Obtaining Updated or Paper-copy Versions of This Document

Silicon Graphics maintains a World Wide Web page from which you can retrieve the latest versions of many of the company's documents, and from which you can obtain instructions for ordering printed (paper-copy) versions of online documents. Using your Web browser, open the following URL:

http://techpubs.sgi.com/library/

To locate the latest versions of IRIS HIPPI documents (including this one), make the following selections:

1. Click on the "Library Search" option.
2. Enter **`hippi`** to search for all titles that contain that string.
3. Click on the document that you want to view, download and print, or purchase in bound hardcopy format.

Chapter 1

What is HIPPI?

This chapter is an introduction to the High-Performance Parallel Interface (HIPPI) protocol, including HIPPI-PH, HIPPI-Serial, and HIPPI-FP. The chapter provides a 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/device. An upper-layer entity (host, network-layer interface, or program) that uses the HIPPI subsystem is sometimes loosely referred to as the source, however, it is more correct to call these software entities source upper-layer protocols (or source ULPs).

It uses *destination* to refer to the receiving endpoint/device. The comment about upper-layer entities, included in the source entry, applies here also.

It uses *fabric* to refer to all the HIPPI nodes (switches, endpoint devices, extenders) that are physically interconnected and speak the same physical-layer protocol. One HIPPI fabric can be logically divided into multiple upper-layer address spaces (that is, networks). For example, a single HIPPI fabric can support multiple IP networks. One network can include members from multiple HIPPI fabrics. For example, an IP network can include members on a HIPPI-PH fabric as well as members on a HIPPI-Serial fabric.

A *word* in the HIPPI environment can be either 4 bytes (32 bits) or 8 bytes (64 bits), depending on the HIPPI implementation. In 800 megabit per second implementations, each word is 4 bytes. 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, point-to-point protocol. HIPPI provides for transmission at 800 or 1600 megabits per second in each direction.[1] Before data can be sent from one HIPPI endpoint to another, there must be both a physical link and a negotiated open connection between them. For its physical layer, a HIPPI implementation can use either the HIPPI-PH or the HIPPI-Serial standard. For HIPPI-PH, each physical link is a copper cable up to 25-meters long. For HIPPI-Serial, each link consists of a fiber-optic cable that can be from 2 to 10,000 meters long (depending on the type of cable and optics).[2] For both HIPPI-PH and HIPPI-Serial, each physical link connects two HIPPI nodes. Each node can be an endpoint or an intermediate HIPPI switch, as illustrated in Figure 1-1, where an endpoint-to-endpoint configuration is illustrated as well as 2 examples of configurations that include switches.

In copper-based implementations, the HIPPI hardware has two 100-pin connectors for connecting copper cables, as illustrated in Figure 1-2; each cable is a single link that carries upper-layer data in one direction and control data in both directions. In HIPPI-Serial implementations, the hardware has 1 connector for a dual-fiber fiber-optic cable, as illustrated in Figure 1-2; the fiber-optic cable contains 2 physical links; each link carries upper-layer and control data in one direction. Some of the control data on each fiber relates to the connection (that is, the data stream) on the other fiber in the same cable. In fiber-based implementations, endpoints and switches must demultiplex the control data that flows in the opposite direction from the data, as illustrated in Figure 1-2.

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

[2] IRIS HIPPI-Serial uses shortwave optics that can support cable lengths of up to 500 meters.

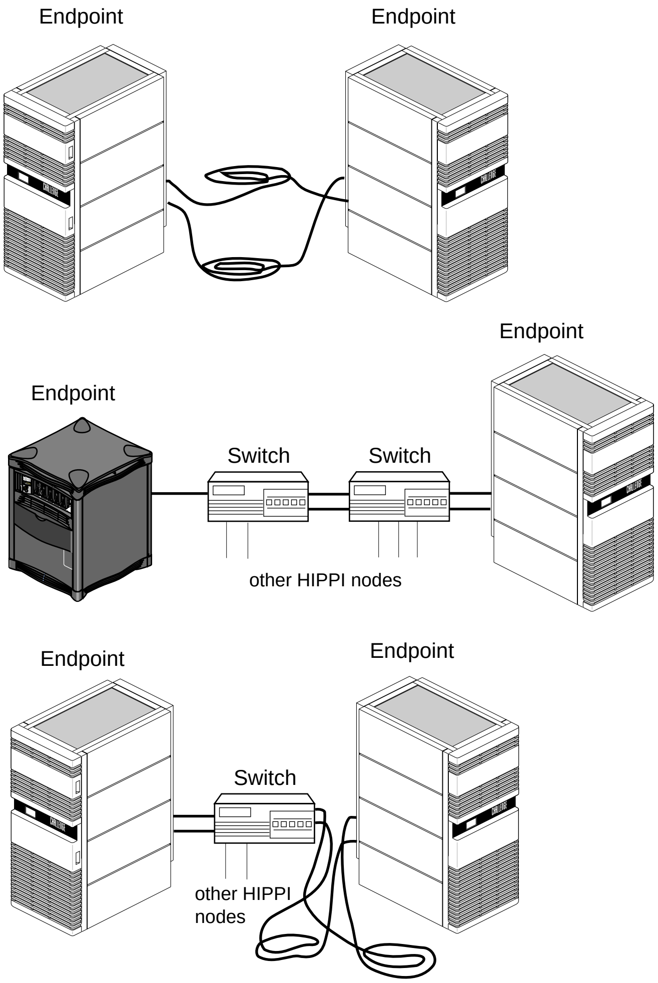


Figure 1-1 Examples of Links Between HIPPI Nodes

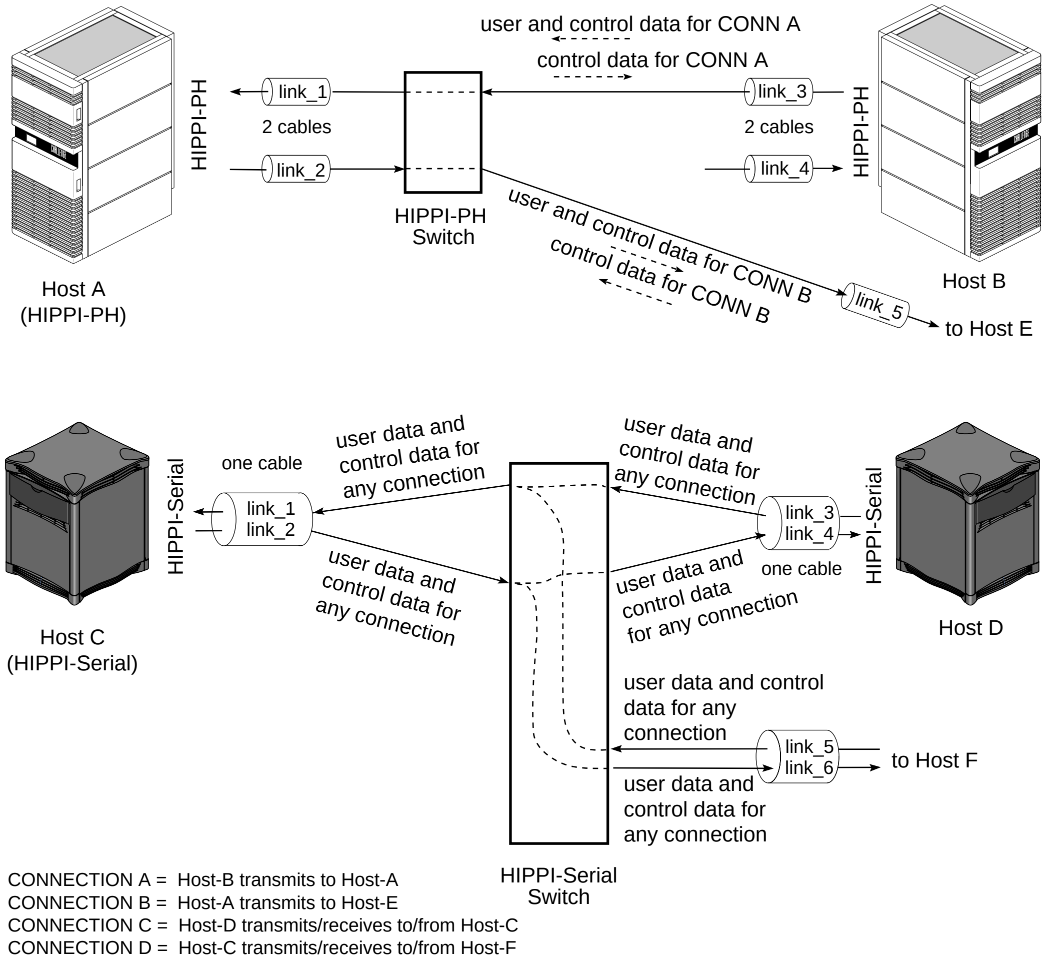

user and control data for CONN A
control data for CONN A
HIPPI-PH
link_1
2 cables
link_2
HIPPI-PH Switch
link_3
2 cables
link_4
HIPPI-PH
Host A (HIPPI-PH)
Host B
user and control data for CONN B
control data for CONN B
link_5
to Host E
one cable
HIPPI-Serial
user data and control data for any connection
link_1
link_2
user data and control data for any connection
user data and control data for any connection
link_3
link_4
HIPPI-Serial
one cable
user data and control data for any connection
Host C (HIPPI-Serial)
Host D
user data and control data for any connection
link_5
link_6
to Host F
user data and control data for any connection
HIPPI-Serial Switch
CONNECTION A = Host-B transmits to Host-A
CONNECTION B = Host-A transmits to Host-E
CONNECTION C = Host-D transmits/receives to/from Host-C
CONNECTION D = Host-C transmits/receives to/from Host-F

Figure 1-2 User and Control Data Carried by HIPPI-PH and HIPPI-Serial Links

The open connection is an agreement for the transfer of upper-layer data from one endpoint to another. To open a connection, the two endpoints exchange HIPPI signals (control data) according to the protocol specified in the HIPPI-PH standard. (HIPPI-Serial uses the same control signals as HIPPI-PH.) In many HIPPI implementations, each link is designed as an independent entity, so that links can each support a connection to a different host, as illustrated by each link_2 example in Figure 1-2.

Note: With HIPPI-PH, Host A can connect directly to both Host-B and Host-E, without the use of a switch in between. With HIPPI-Serial, Host-C can connect directly to Host-D without a switch; however, for Host-C to connect to Host-D and Host-F, as illustrated, an intermediate switch (or similar device for multiplexing the control data) must be used.

Figure 1-3 illustrates a configuration of HIPPI equipment with six different physical links and nine possible endpoint-to-endpoint connections (listed below), of which three can be simultaneously active (engaged in open connections):

- A-source transmitting (over link_1) to any one of the following:
    - A-destination (itself, over link_2)
    - B-destination (link_4)
    - C-destination (link_6)
- B-source transmitting (over link_3) to A-destination (link_2), B-destination (itself, over link_4), or C-destination (link_6)
- C-source transmitting (over link_5) to A-destination (link_2), B-destination (link_4), or C-destination (itself, over link_6)

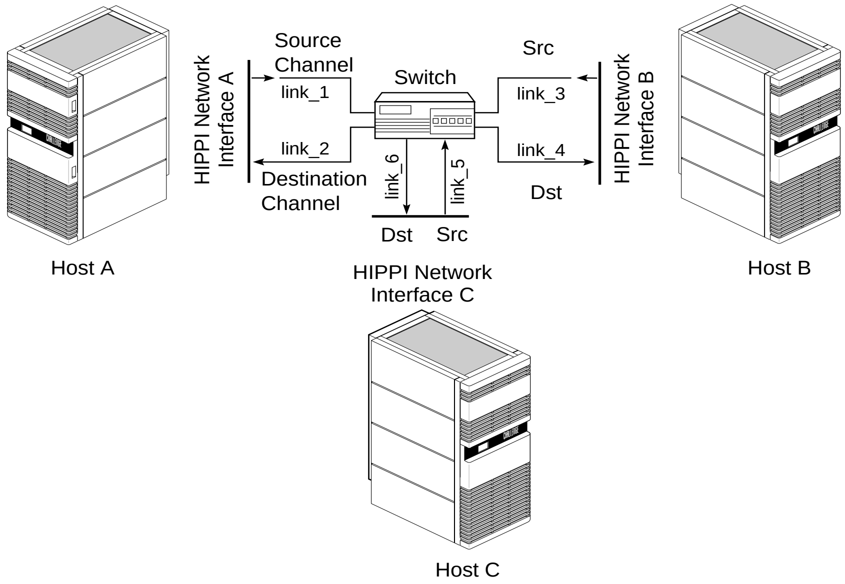


Figure 1-3 HIPPI Links and Connections

An open connection consists of an exchange of signals (control data) between a source and a destination. (Figure 1-19 illustrates the signals.) During this exchange, the destination agrees to accept data exclusively from the source; throughout the transfer of the upper-layer data, the destination uses backflowing signals to inform the source of its ability to accept more data. Each link supports only one connection (that is, HIPPI is point-to-point). To move data in both directions between two hosts, two endpoint-to-endpoint links (or series of 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 physical link (or links) between the two HIPPI interfaces contains data packets transmitted only by the source (that is, HIPPI connections are simplex). HIPPI packets may be seen by intermediate switches but not by other host interfaces. A connection may be kept open for extended periods of time, even when there is no data moving across it, or it may be closed by either endpoint at any time; 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 an incoming and an outgoing **INTERCONNECT** signal. 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 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" on page 22). The I-field contains (among other things) routing information, used by switches 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 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. At least one of the physical links on route 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 interface is usually the one to do this.

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.[1] 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 1-4. 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).[2] At the end of each

[1] The source channel on the IRIS HIPPI for Challenge/Onyx board can enqueue up to 65,535 **READY**s; the destination channel can generate up to 255 outstanding **READY**s. Each channel of the IRIS HIPPI-Serial XIO board can handle 128 **READY**s.

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

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.

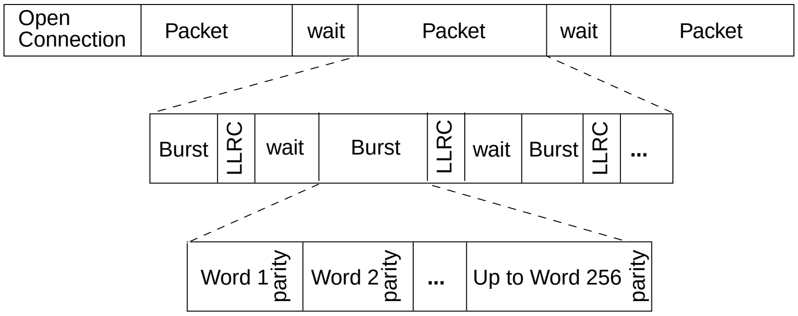


Figure 1-4 HIPPI Packets and Bursts

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" on page 23). 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.

HIPPI MAC-Level 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 fabric, thus implementing 2 different HIPPI address spaces (networks).

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 1-5. The order in which the addresses are placed within the field is defined by the I-field’s Direction bit, as illustrated in Figure 1-5.

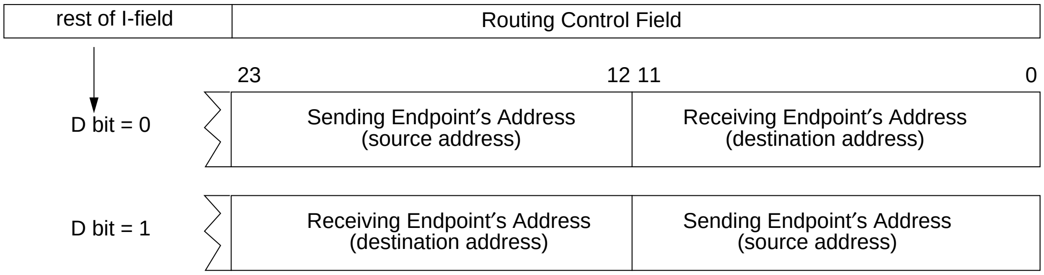

rest of I-field
Routing Control Field
23
12 11
0
D bit = 0
Sending Endpoint’s Address (source address)
Receiving Endpoint’s Address (destination address)
D bit = 1
Receiving Endpoint’s Address (destination address)
Sending Endpoint’s Address (source address)

Figure 1-5 Routing Control Field With Logical Addressing

When a network uses HIPPI logical addressing, each HIPPI endpoint within the network is assigned an address that is unique within that network; each address must also be unique within all the HIPPI fabrics across which the network extends. In most configurations (and in all HIPPI-Serial configurations), one address is used for both the source and destination channels of each endpoint. Assignment of these addresses is a local matter; the addresses do not need to be unique outside the HIPPI fabrics and networks that are involved.

Although each HIPPI address identifies one member of the network, the address is configured at the switch (not at the endpoint). Each address is mapped to the port (on the switch) to which that endpoint is attached. (The exact method for configuring addresses to ports is different for each switch vendor.) Each upper-layer protocol (ULP) module must learn the address that has been assigned to its own HIPPI subsystem’s link, as well as the address assigned to each destination’s link. The method for discovering these addresses is defined at the upper-layer protocol level (for example, by the HIPPI-LE standard). For IP implementations that are conformant with RFC 1374, a static (table lookup) address resolution scheme can be used to inform the IP stack of the local and destination HIPPI addresses. The system administrator may need to create the IP “ARP” table manually. Dynamic address resolution (ARP) is also supported as described in the section “IP Address Resolution with HARP” on page 18.

Logical addresses have the formats described in Table 1-1. Some of the addresses are reserved for special purposes.

Table 1-1 Logical Address 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 logical HIPPI network and uses a routing table to select the path along which to open a connection for each request. For example, Figure 1-6 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.

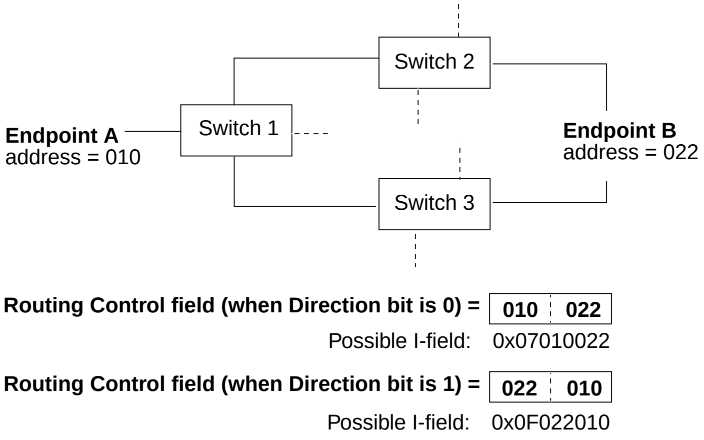

Switch 2
Endpoint A
address = 010
Switch 1
Endpoint B
address = 022
Switch 3
Routing Control field (when Direction bit is 0) =
010
022
Possible I-field: 0x07010022
Routing Control field (when Direction bit is 1) =
022
010
Possible I-field: 0x0F022010

Figure 1-6 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 endpoints. 4032 is the maximum number of destination endpoints that can exist within one HIPPI network using logical addressing.

Source Addressing

Note: If you use source addressing and more than one switch you must use static address resolution as described in "/usr/etc/hippi.imap File" on page 59.

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 1-7. 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 1-7.

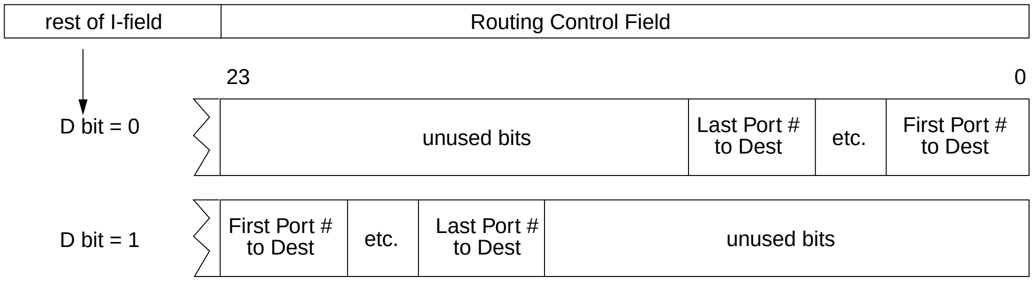


Figure 1-7 Routing Control Field (As Created by Sender) Source Addressing

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 1-8. Port identifiers are unique among all the ports on the same switch, but not among all the ports within the network or fabric. For example, a network/fabric with 5 switches might easily have 5 port identifiers of "1." Figure 1-8 is an example of the port identifiers used in a network with 2 switches.

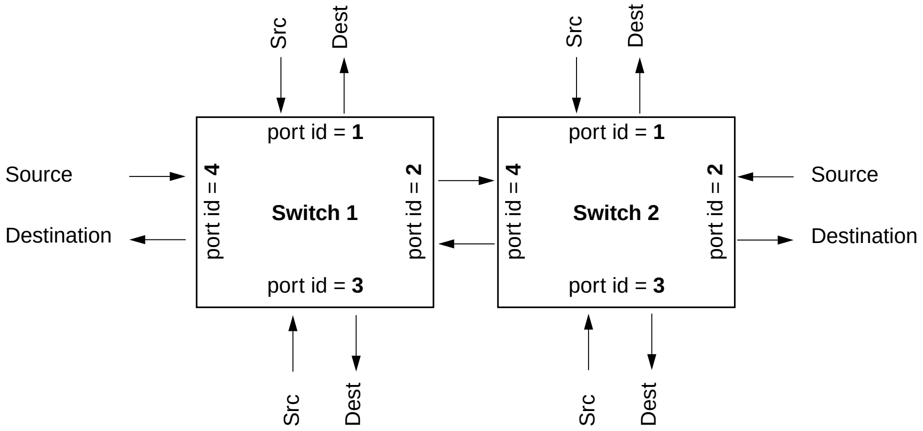


Figure 1-8 Switches and Port Identifiers

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 (source 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 endpoints, the address consists of one port identifier: the one to which the receiving interface is connected, as illustrated by Example 1 in Figure 1-9. When two switches exist between the interfaces, the address consists of two port identifiers, as illustrated by Example 2 of Figure 1-9.

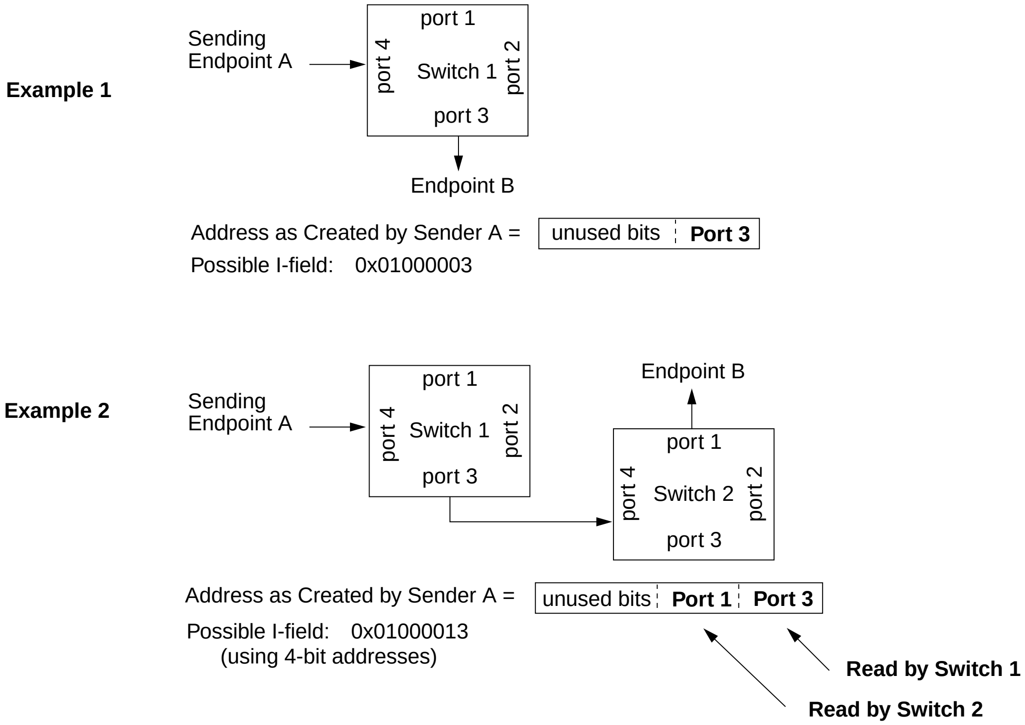
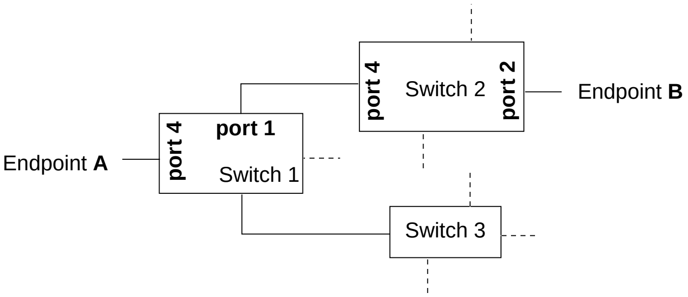


Figure 1-9 Port Identifiers for Source Addressing

Each endpoint application must learn the address (series of port identifiers) required to reach each destination's link. (Note that with source addressing, an endpoint does not need to know its own address.) The method for discovering these addresses is defined at the upper-layer level (for example, HIPPI-LE), so it varies depending on the ULP. For example, for IP implementations that are conformant with RFC 1374, a static (table lookup) address resolution scheme can be used to inform the IP stack of the HIPPI addresses. The table is manually created by a system administrator who gathers the necessary information from each switch and creates the addresses between the different endpoints. You can also use dynamic address resolution protocol (ARP) as described in the section "IP Address Resolution with HARP" on page 18.

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 1-10 illustrates two addresses that endpoint A might use to open a connection with B.



Routing Control Field as created by sender (when D bit is 0) = | unused | **2** | **1** |
Possible I-field: 0x01000021 (using 4-bit addresses)

Routing Control Field as created by sender (when D bit is 1) = | **1** | **2** | unused |
Possible I-field: 0x09120000 (using 4-bit addresses)

Figure 1-10 Routing With Source Addressing

Each HIPPI upper-layer protocol (ULP) that uses the HIPPI network 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 ULP 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 on route to the destination), addresses in source addressing format are changed by each switch that handles the I-field. The source ULP 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 1-11.

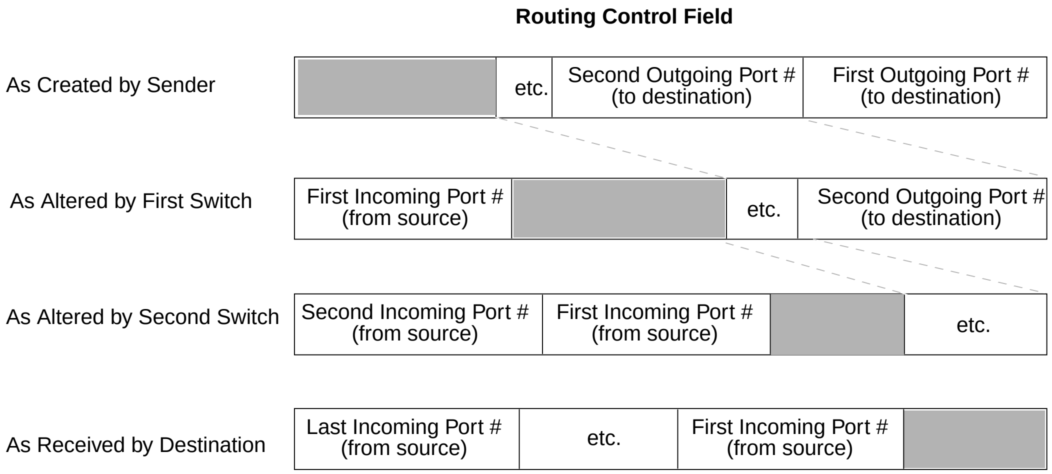


Figure 1-11 How Switches Alter Source Addresses

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 1-11 becomes the *First Outgoing Port #* for the return connection.

Port identifiers can be one to six bits in length. 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 2-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 network'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 Summary of Routing Control Field Bits

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

IP Address Resolution with HARP

The address resolution protocol for HIPPI networks (including HIPPI-800) is specified in */internet-drafts/draft-pittet-hippiarp-02.txt* at *ftp.ietf.org*. HARP provides a dynamic, client/server-based address resolution service. The protocol makes it possible for each IP endpoint (client) on the network to register its own INET and HIPPI hardware addresses with the server and to discover the HIPPI hardware address for other known INET addresses. An INET address can be known, for example, from the site's NIS or */etc/hosts* database. The HIPPI hardware address (HWA) is the I-field.

The HARP server maintains a kernel-resident lookup table (database) that maps INET addresses to HIPPI hardware addresses. HARP occurs in 2 phases: a registration phase (summarized in "HARP Registration Phase" on page 19) and a normal operation phase (summarized in "HARP Operation Phase" on page 20).

When a HIPPI fabric includes one or more endpoints that do not support dynamic HARP, static mappings for those endpoints must be added. Silicon Graphics recommends you add these static addresses to the HARP server's */usr/etc/hippi.imap* file. See "/usr/etc/hippi.imap File" on page 59 for more information.

HARP Registration Phase

During startup, each dynamic HARP client registers its address pair (INET and HIPPI hardware address) with the HARP server. The client does this by sending an inverse address resolution request (InARP_Request) to the HARP server. The InARP_Request contains the client's INET address and HIPPI hardware address; the request asks for the server's INET address.

The client must know both its own HIPPI hardware address, and the address of the server (you must configure this information manually on each client as described in "/usr/etc/ifhip.conf File" on page 56). If no reply comes back to the client, the client waits 5 seconds and tries again, until it succeeds. If the system has been configured with a backup HARP server, the client sends its retry request to that alternate address.

When the server receives an InARP_Request, it enters the new INET-to-HIPPI hardware address mapping into its table and replies with an InARP_Reply message that provides its own INET address to the client.

Figure 1-12 illustrates the exchange that occurs between client and server during registration.

The server keeps each registered entry in its database for a maximum of 20 minutes. Whenever an entry has aged beyond this maximum, the server removes the entry. To prevent its entry from disappearing, each functional client re-registers itself with the server at least once every 15 minutes.

Note: When an endpoint is engaged in a very large data transaction, the HARP client at that endpoint cannot access the link to communicate with its HARP server. This may cause the client's entry in the HARP server's database to disappear.

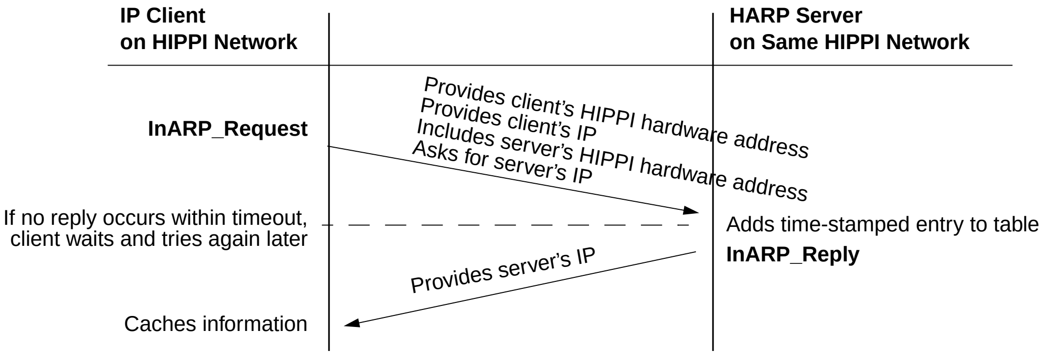

IP Client
on HIPPI Network
HARP Server
on Same HIPPI Network
InARP_Request
Provides client's HIPPI hardware address
Provides client's IP
Includes server's HIPPI hardware address
Asks for server's IP
If no reply occurs within timeout,
client waits and tries again later
Adds time-stamped entry to table
InARP_Reply
Provides server's IP
Caches information

Figure 1-12 HARP Registration

HARP Operation Phase

During normal operation, each HARP client requests address resolution from the server for each IP destination. The client makes this request by sending an ARP_Request to the server; the request contains the desired destination's (that is, target's) INET address.

The server looks up the INET address in its HARP table to discover the HIPPI hardware address, then responds with an ARP_Reply that supplies the target's HIPPI hardware address. If the server does not find a mapping for the INET address, it responds with an ARP_NAK. Figure 1-13 illustrates the message exchange that occurs between client and server during normal operation.

An ARP_Reply indicates one of the following:

- The targeted client supports dynamic HARP and within the last 20 minutes was functional enough to register itself with the server.
- The targeted client does not support dynamic HARP and its entry at the server is a static entry. There is no way to know whether this client is functional.

An ARP_NAK indicates that the IP client (endpoint) has not registered itself with the server for at least 20 minutes. This failure to register can be caused by any of the following conditions:

- The endpoint is powered off.

- The link to the client has been unavailable for 20 minutes or more. For example, the link might be occupied by a non-IP protocol stack or the cable might be loose.
- The HARP client software at the endpoint is not up and running.
- The endpoint does not support dynamic HARP and no one has added a static entry for it to the HARP server′s database.

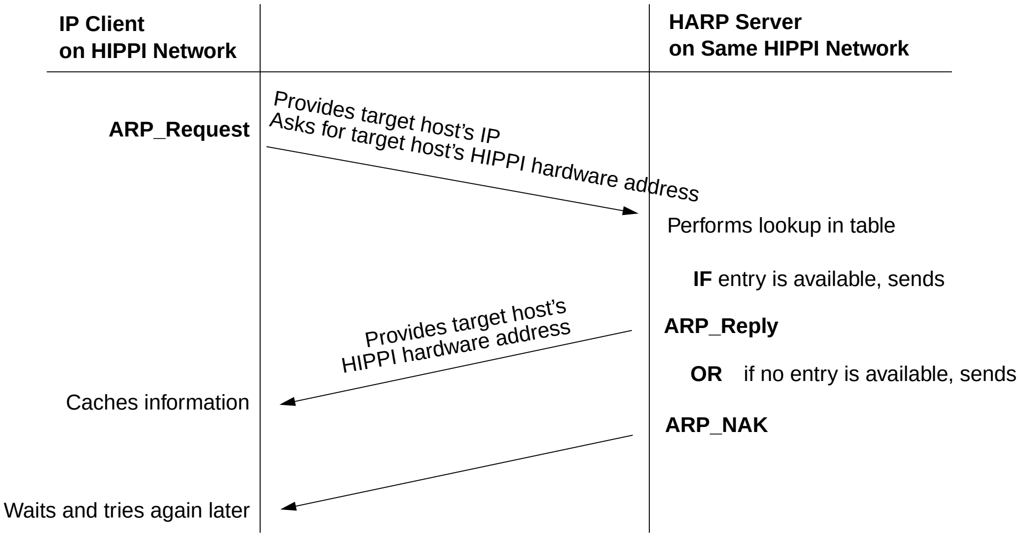


Figure 1-13 HARP Normal Operation

HARP Broadcast Emulation

HARP provides software emulation of IP broadcasting by replicating packets and sending them to each host. This may require significant network bandwidth because the bandwidth is effectively divided by the number of HARP clients being served.

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 1-14, and its fields are explained in Table 1-3.

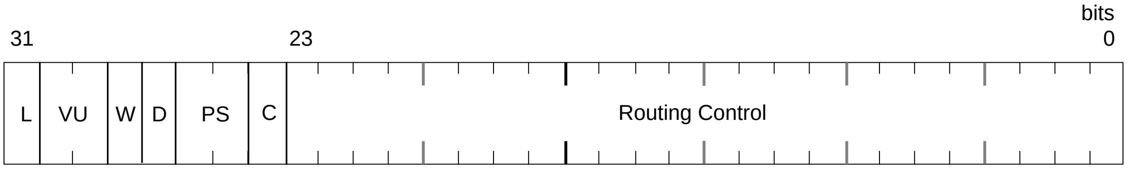


Figure 1-14 I-field Format

Table 1-3 Summary of I Field Bits

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.

Table 1-3 (continued) Summary of I Field Bits

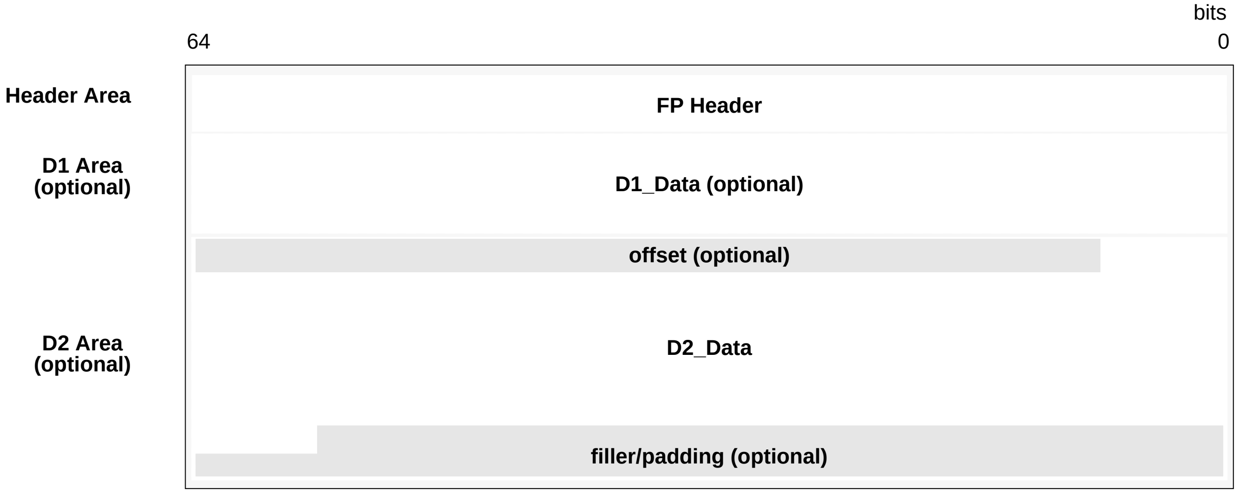
Field	Bits	Description
D	27	Direction: 0=Least significant bits of Routing Control field contain the destination address for the current switch to use. 1=Most significant bits of Routing Control field contain the destination address for the current switch to use.
PS	26:25	Path Selection: 00=Source routing. 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.
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 24-bit 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 1-15. 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.
- 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 1-15 HIPPI-FP Packet Format

The 64-bit FP header describes the HIPPI packet using six fields, illustrated in Figure 1-16 and described in Table 1-4.

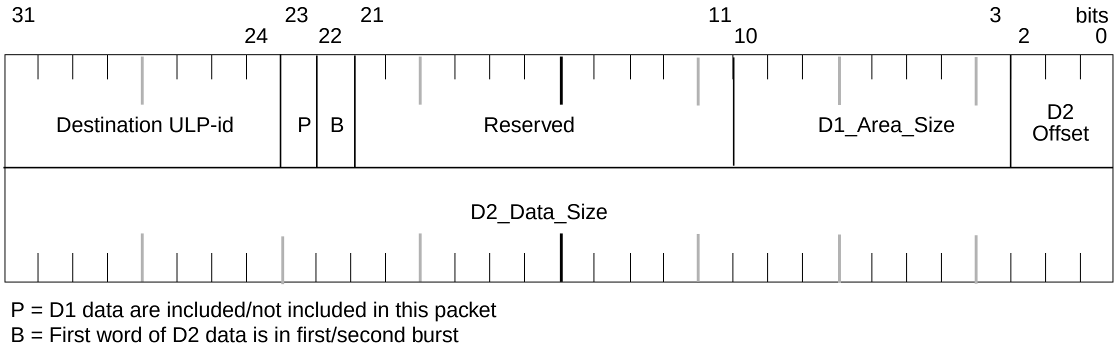

31
24
23
22
21
11
10
3
2
bits
0
Destination ULP-id
P
B
Reserved
D1_Area_Size
D2 Offset
D2_Data_Size
P = D1 data are included/not included in this packet
B = First word of D2 data is in first/second burst

Figure 1-16 FP Header Format

Table 1-4 Summary of FP Header Bits

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 (defined by the D1_Area Size field), but empty (the P bit is set to 0). 0=no D1 data for this packet; 1=D1 data exists for this packet.
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. 0=there is D2 data in the packet's first burst; 1=D2 data starts on first byte of second burst.

Table 1-4 (continued) Summary of FP Header Bits

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. This field is used only when D2 data is present in the first burst.
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 1-17 illustrates a HIPPI-FP packet. The first burst of this packet (words 0 to 255) contains only the FP header and the D1 data. The D1 area in this packet is completely full of D1 data. Notice that the P bit in the FP header is set to 1 to indicate the presence of valid D1 data. The D2 data starts on word 256, the first byte of the second burst, as indicated by the B bit setting.

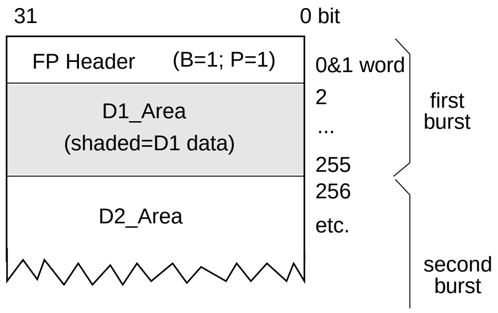


Figure 1-17 Sample HIPPI-FP Packet

Figure 1-18 illustrates some of the HIPPI-FP packets that are commonly created by applications. Notice how example 2 uses an empty D1_Area to pad out the first burst, thus locating the data in the second burst. Examples 2, 3, and 4 all illustrate this technique. Also notice, examples 1 and 5, how user (D2) data can be placed in the first burst. with or without D1 data.

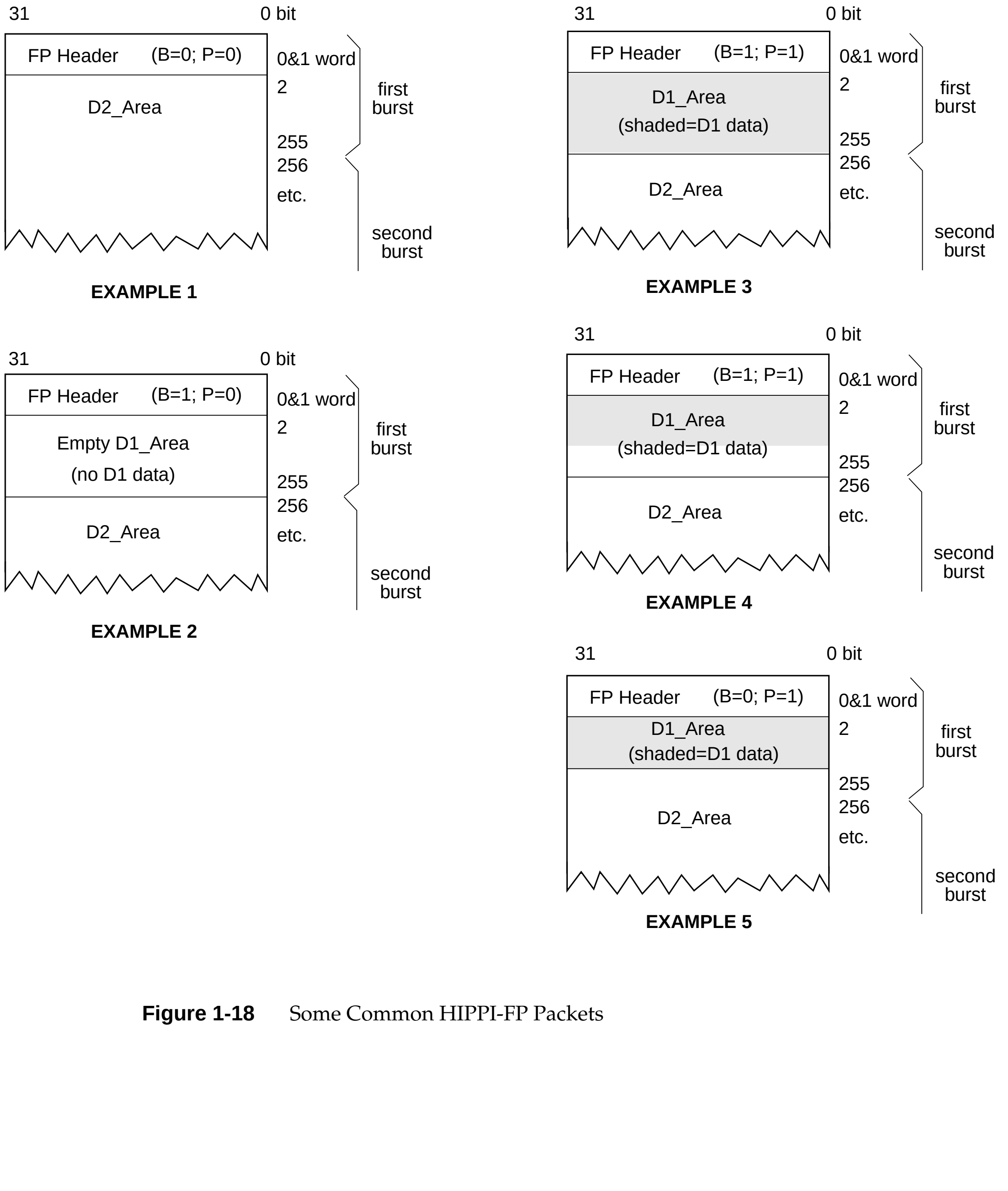


Figure 1-18 Some Common HIPPI-FP Packets

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 1-19. 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.

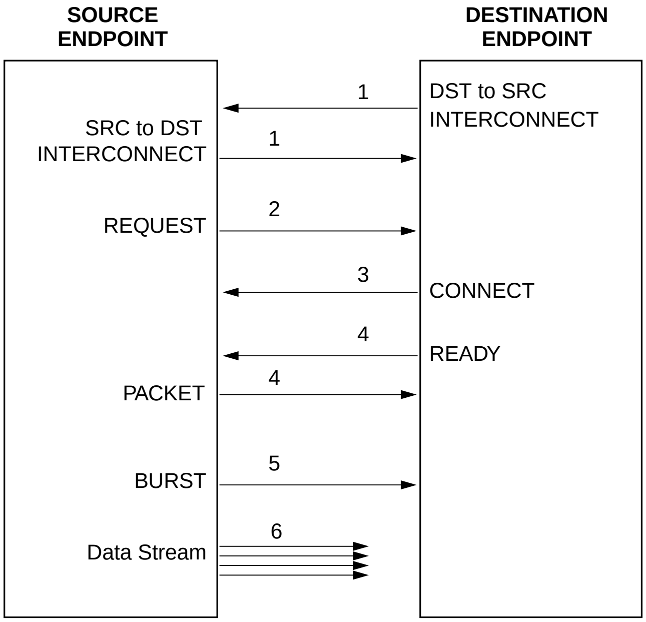


Figure 1-19 HIPPI Signals Used on Each Point-to-Point Connection

Table 1-5 HIPPI-PH Signal Summary

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**.
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 Configurations

This section describes some of the common configurations of HIPPI equipment. Because HIPPI is a simplex point-to-point protocol (or in the case of HIPPI-Serial, dual-simplex), only one source can transmit user (upper-layer) data onto the transport medium (cable) between the two endpoints. Two physical links 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 HIPPI configuration consists of two endpoints, one sending and the other receiving user data, as shown in Figure 1-20.

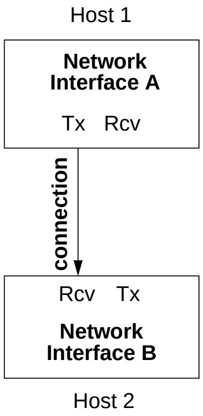


Figure 1-20 Basic HIPPI Configuration

To exchange data in both directions, two physical links and two connections are required between the two endpoints.[1] Each endpoint's source channel must open a connection with the destination of the other endpoint. Some HIPPI products (for example, HIPPI from Silicon Graphics, but not serial HIPPI) allow each simplex connection to connect to a different endpoint, as illustrated by the two examples on the right in Figure 1-21.

[1] In HIPPI-Serial implementations, each fiber-optic cable attached to a dual-SC port contains 2 links.

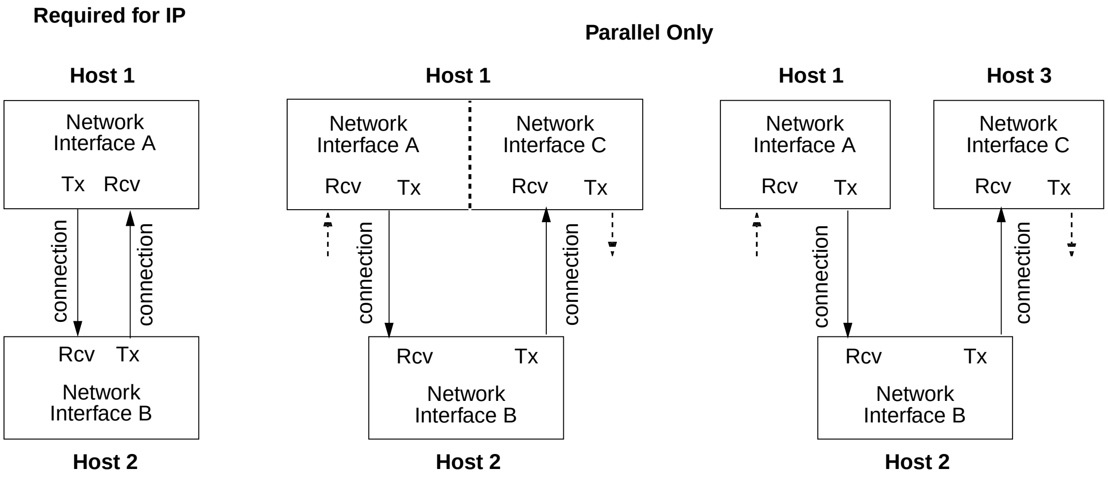


Figure 1-21 Three Variations of the Basic Configuration

The copper-based IRIS HIPPI mezzanine board for the CHALLENGE and Onyx platforms has two ports: one provides the source physical link and the other provides the destination physical link. The fiber-optics based IRIS HIPPI-Serial XIO board for the Origin series and Onyx2 platforms has one port that supports the two simultaneous connections in opposite directions. The IRIS HIPPI software treats each link as a separate entity, so that each IRIS HIPPI board can support two autonomous, simultaneous connections (one sending and one receiving). The two connections can be to two different endpoints (as shown by the two examples on the right in Figure 1-21) or to the same endpoint (as illustrated by the example on the left in Figure 1-21). IP communication over HIPPI requires the latter configuration.

HIPPI Fabric 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 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 that may include demultiplexing functionality; but they are 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 1-22 illustrates a HIPPI fabric with one switch. The switch in this illustration is a 4 x 4, meaning that the switch can have four systems (8 HIPPI links) attached to it. The switch supports four simultaneous connections. For example, in Figure 1-22, 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 B 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.

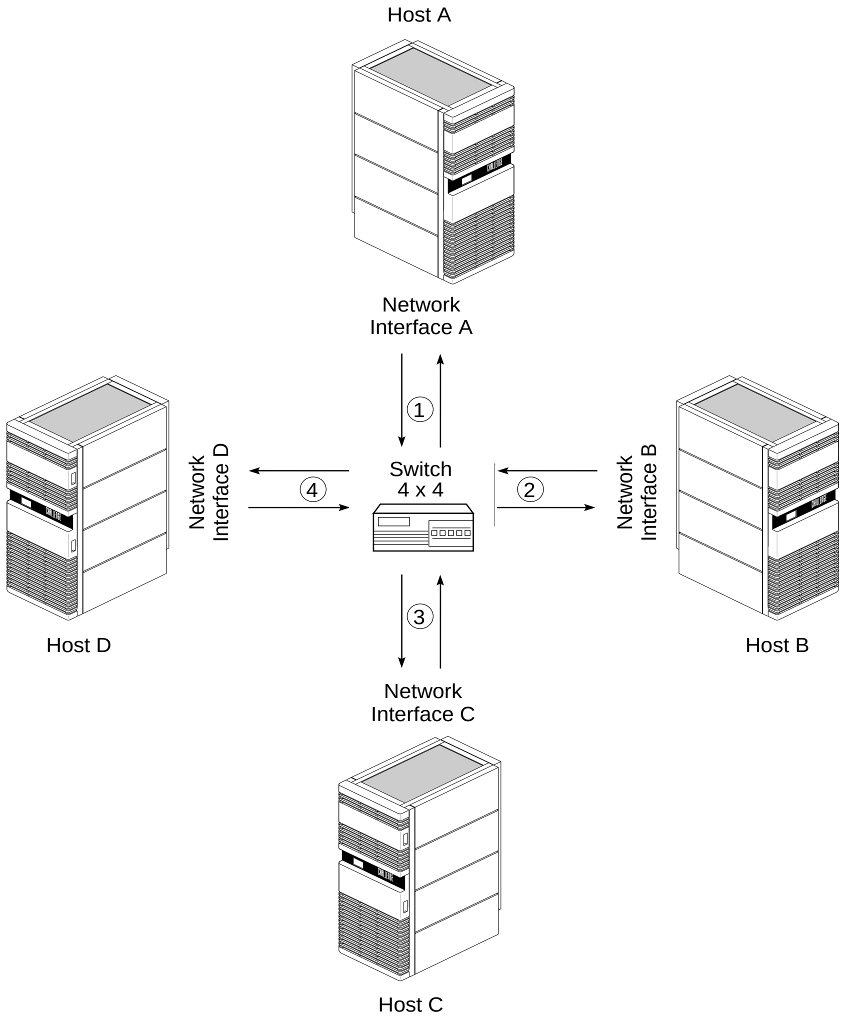


Figure 1-22 HIPPI Fabric Configuration With One Switch

Figure 1-23 illustrates a fabric with multiple switches, and Figure 1-24 illustrates a complex HIPPI fabric 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.

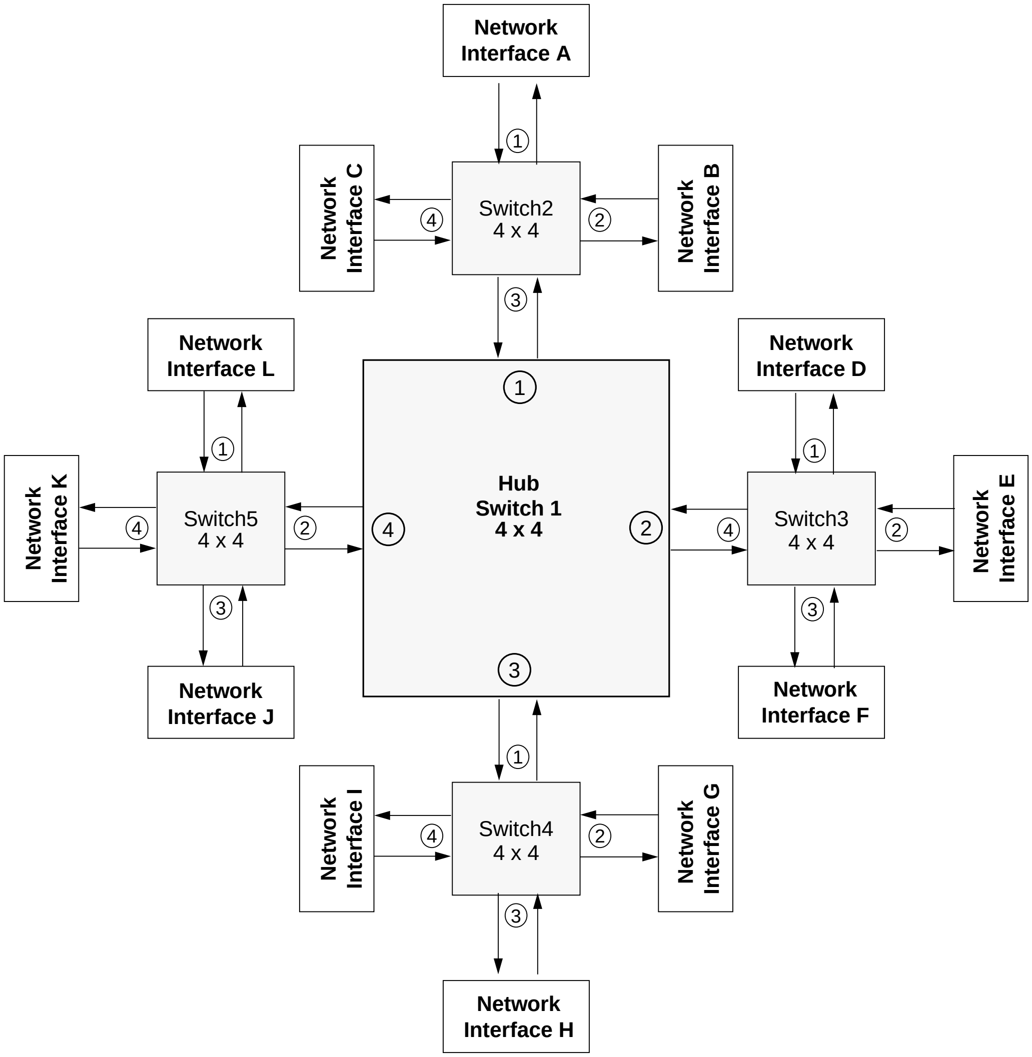


Figure 1-23 HIPPI Fabric Configurations With Multiple Switches

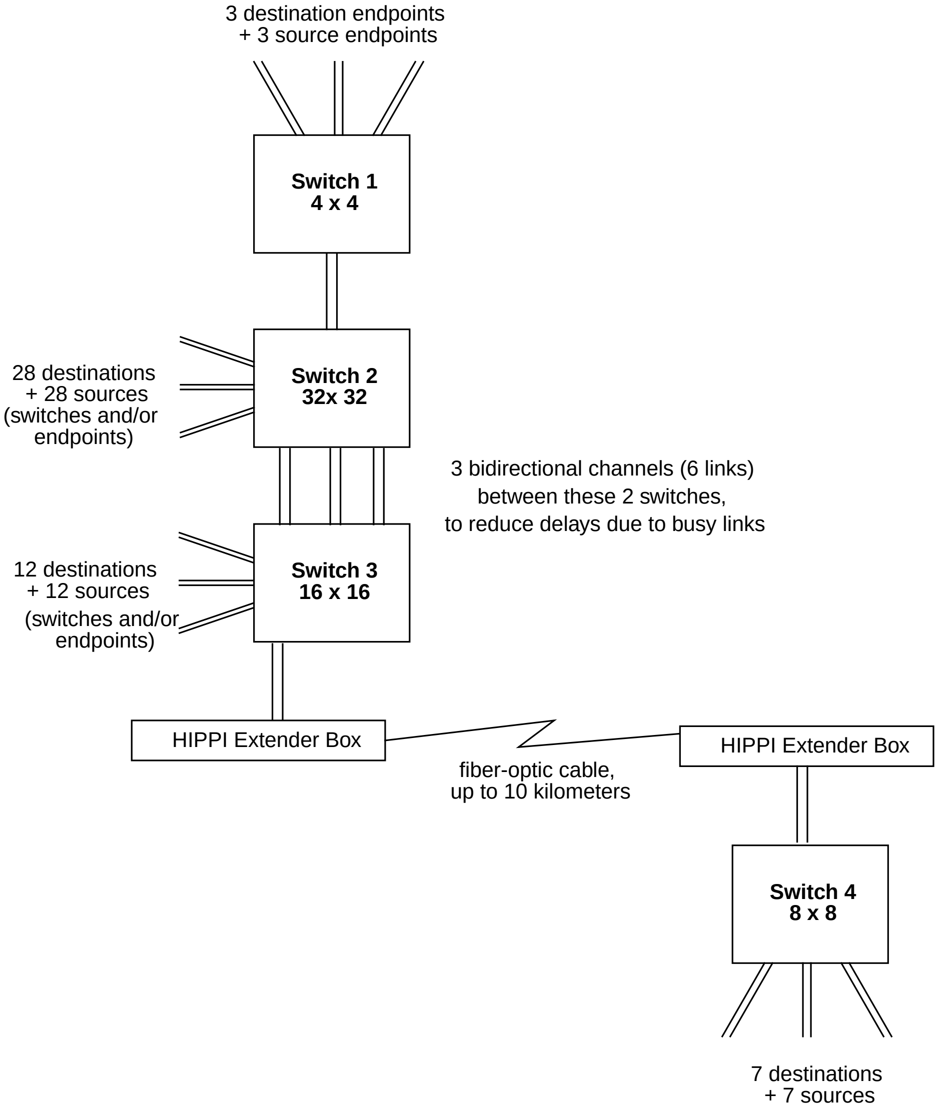


Figure 1-24 Complex HIPPI Fabric Configuration

The maximum number of switches and endpoints in a network is limited by 3 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 network. 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" on page 13. 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 network, 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 network is limited. This limit affects the configuration of the network.
- If a network 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 network. 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 1-23 shows an example of an RFC 1374-compliant network. Table 1-7 summarizes the maximum number of switches and endpoints possible for a network in which switches of only one size are used throughout the network.

Table 1-6 Source Routing Path Switch Totals

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 Recommended Maximum Switches Per Appendix B of RFC 1374

Size of All Switches Within Network				
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 official definitions for HIPPI and how it works.

- ANSI HIPPI-PH
  The *HIPPI Mechanical, Electrical, and Signalling* document defines the standard for a copper physical layer: electrical and mechanical aspects of copper HIPPI cables, connectors, transmitters, and receivers. This document defines the HIPPI signals (like **SDIC, DSIC, REQUEST**, **CONNECT, READY**, **PACKET**, and **BURST**) that are used in all HIPPI implementations, including HIPPI-Serial.
- ANSI HIPPI-Serial
  The *High-Performance Parallel Interface-Serial Specification: X3.300-1997* document defines the ANSI standard for a fiber-optic physical layer: electrical and mechanical aspects of fiber-optic cables, connectors, transmitters, and receivers. The method for communicating (encoding) the HIPPI-PH signals in serial is also defined.
- ANSI HIPPI-SC
  The *HIPPI Physical Switch Control* document defines standards for switch behavior, routing methods, and connection management. It also defines the HIPPI I-field.
- 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). This standard also defines a dynamic address discovery handshake for endpoints to use with their attached switches.

- 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 2067
  IP over HIPPI, by J. Renwick (January 1997) defines the protocols for using the IP suite of network and transport layer protocols over HIPPI.
- Draft of Proposed HARP Standard
  The internet draft by J.-M. Pittet, February 1999, defining the HARP interface is available over *ftp* from *ftp.ietf.org*. at */internet-drafts/draft-pittet-hippiarp-02.txt*.
- RFC 1323
  TCP Extensions for High Performance, by V. Jacobson, R. Braden, and D. Borman (May 1992) defines the protocol for scaled windows and timestamps designed to improve performance for large bandwidth and very high-speed products.

The ANSI documentation for HIPPI standards is maintained by the American National Standard of Accredited Standards Committee (ANSI X3T9.3). Information about HIPPI standards is available on the HIPPI Network Forum’s webpage *http://www.hnf.org*, and many HIPPI documents are posted on the HIPPI summary webpage *http://www.hippi.org*. Standards documents are available by phoning ANSI at +1-212-642-4900.

Implementation Details for IRIS HIPPI

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

How HIPPI Ports Are Assigned to IP 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 port.

On CHALLENGE and Onyx Platforms

On Silicon Graphics systems that have HIO slots (for example, CHALLENGE, Power Challenge, Onyx, or Power Onyx systems), with each restart, the startup routine probes for hardware installed in the mezzanine I/O adapter slots and makes a list (inventory) 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 ports 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 port located and `hippi1` is the second. In this example, the startup routine located two IRIS HIPPI ports attached to FMezz boards on two different IO4 boards.

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

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

- If the IRIS HIPPI driver is configured to support the IP protocol stack, the driver creates a network interface for each HIPPI port in the inventory. The first HIPPI interface (always named *hip0*) is associated with the first port listed in the inventory; the second interface (*hip1*) is associated with the second port on the list; and so on, until there are no more ports. For example, using the configuration shown in the *hinv* example above, interface *hip0* is assigned to port *hippi0* and *hip1* is assigned to port *hippi1*.
- The *ifconfig* command (which is invoked automatically during startup) searches the *netif.options* file for IP-over-HIPPI interface names (for example, *hip0*, *hip1*, *hip2*) and configures and enables each interface that exists (that is, each interface that was created by the driver).

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 ports may change. For example, *hip0* (from the example above) could be assigned, at a later reboot of the machine, to the *hippi1* port instead of *hippi0,* if *hippi0* were not found.

On Origin and Onyx2 Platforms

On an Origin series or Onyx2 system, with each restart (for example, after a *reboot*, *shutdown*, *halt*, *init* command, or a power off), the startup routine probes for hardware on all the systems connected into the CrayLink interconnection fabric. All the slots and links in all the modules within the fabric are probed. The routine then creates a hierarchical file system, called the hardware graph, that lists all the hardware that is located. The top of the hardware graph is visible at */hw.* (For complete details, see the hwgraph(4) reference page.) After the hardware graph is completed, the *ioconfig* program assigns a unit number to each located device that needs one. Other programs (for example, *hinv* and the device's driver) read the assigned number from *ioconfig* and use it. On an initial startup, *ioconfig* assigns numbers sequentially; for example, if three IRIS HIPPI boards are found, they are numbered *unit0*, *unit1*, and *unit2*. On subsequent startups, *ioconfig* distinguishes between hardware that it has seen before and new items. To previously seen items, it assigns the same unit number that was assigned on the initial startup. To new hardware, it assigns new sequential numbers. It never reassigns a number, even if the device that had the number is removed and leaves a gap in the numbering.

Note: New items are differentiated from previously-seen items based solely on the hardware graph listing (that is, the path within */hw*). The database of previously-seen devices is kept in the file */etc/ioconfig.conf*. For example, a new replacement board that is installed into the location of an old board will be assigned the old board's number, while a board that is moved from one location to another will be assigned a new number. For more information about the hardware graph and *ioconfig*, see the reference (man) pages for *hwgraph* and *ioconfig*.

The IRIS HIPPI boards that are located can be displayed with the */sbin/hinv* or *find* commands, as shown below. In these examples, the startup routine located two IRIS HIPPI boards on two different modules (that is, inside two different chassis).

```
% find /hw/module -name hippi
/hw/module/1/slot/io3/hippi_s/pci/0/hippi
/hw/module/2/slot/io12/hippi_s/pci/0/hippi
```

```
% /sbin/hinv -d hippi
HIPPI-Serial adapter: unit 0, in module 1 I/O slot 3
HIPPI-Serial adapter: unit 1, in module 2 I/O slot 12
```

As the startup process continues, it calls the network hardware drivers to create their network and programmatic interfaces. For HIPPI, this step works in the following way:

- For each IRIS HIPPI board, the installation script creates a symbolic link in */dev* that points to the board's entry in the hardware graph. Subsequently, the driver creates short (*/hw/hippi/#*) and long (*/hw/module/#/slot/.../hippi.*) entries in the hardware graph. The */dev/hippi#* links are for use by the IRIS HIPPI application programming interface (API).
- If the driver is configured to support the IP protocol stack, the driver creates an IP network interface for each board. The network interface number always matches the board's assigned unit number. For example, if the only IRIS HIPPI board found during startup is known by *ioconfig* as *unit2*, then the driver creates only network interface *hip2*.
- The *ifconfig* command (which is invoked automatically during startup) searches the *netif.options* file for IP-over-HIPPI interface names (for example, *hip0*, *hip1*, *hip2*) and configures and enables each one that has been created.

Note: The assignment of network interfaces to boards does not change across restarts. If a board that was once known to the system is not located during a reboot, the network interface that matches that board is not created. For example, if IRIS HIPPI *unit 1* is not found, the *hip1* network interface is not created; the *unit2* and *hip2* pairing proceeds as normal.

Note: If you are using OS Bypass functionality, a maximum of 16 OS Bypass devices are supported, where the device numbers must be between zero and 15. If you are using the character device driver interface, a maximum of 24 devices are supported.

Site Cabling

IRIS HIPPI for Challenge and Onyx platforms is copper based (HIPPI-PH). Each I/O panel plate has two 100-pin connectors. For each IRIS HIPPI board, the site must provide two cables of up to 25 meters each.

IRIS HIPPI for Origin and Onyx2 platforms is fiber-optic based (HIPPI-Serial). Each I/O panel plate has one dual-SC receptacle. For each IRIS HIPPI board, the site must provide one fiber-optic cable of either 50-micron core (3 to 500 meters in length) or 62.5-micron core (3 to 200 meters). The IRIS HIPPI-Serial board uses short wavelength (770-860 nanometer) optics. The cable should be terminated with a dual-SC connector; however, 2 simplex SC connectors will also work. (Figure 1-25 illustrates the data direction used in the board's dual-SC receptacle.) Each cable must conform with the HIPPI-Serial physical layer specification and guidelines, as described in the *ANSI High-Performance Parallel Interface-Serial Specification*, version 2.6, especially annex B and the section entitled "Serial Optical Interface." 62.5-micron core cables in various lengths (including a loopback cable for testing) can be purchased from Silicon Graphics, as summarized in Table 1-8. In addition, the site cabling throughout the HIPPI switch fabric must conform to this specification. Table 1-9 summarizes some of this specification's information.

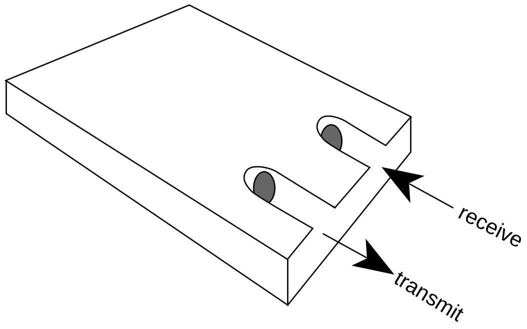


Figure 1-25 Data Direction Used in IRIS HIPPI-Serial Dual-SC Receptacle

Table 1-8 Silicon Graphics 62.5-micron Core Cable Lengths

Description	Part Number
3-Meter Fiber Optic Cable Assembly	018-0656-001, X-F-OPT-3M
10-Meter Fiber Optic Cable Assembly	018-0656-101, X-F-OPT-10M
25-Meter Fiber Optic Cable Assembly	018-0656-201, X-F-OPT-25M
100-Meter Fiber Optic Cable Assembly	018-0656-301, X-F-OPT-100M
Fiber Optic Loopback Test Cable Assembly	018-0664-001, X-F-OPT-LOOP

Table 1-9 Some ANSI-standard Cable Specifications

Item	Value for 50 micron cable	Value for 62.5 micron cable
Cable		
minimum bandwidth at 780 nm	500 MHz•km	160 MHz•km
maximum loss	4 dB/km	4 dB/km
Connectors		
maximum number between 2 nodes	8	8
mean loss per unit	≤0.11 dB	≤0.11 dB
standard deviation loss per unit	0.15 dB	0.15 dB
minimum optical return loss	20 dB	20 dB
Splices		
maximum number between 2 nodes	6	6
mean loss per unit	≤0.08 dB	≤0.08 dB
standard deviation loss per unit	0.05 dB	0.05 dB

Application Programming Interface

The IRIS HIPPI driver provides access and control of the IRIS HIPPI subsystem to upper-layer protocol modules (ULPs). The ULPs that are shipped with the IRIS HIPPI product are the IRIS HIPPI-LE module serving the IP network stack. ULPs can also be developed by customers, using the IRIS HIPPI application programming interfaces.

Customer-developed applications can define their own ULP modules and use the IRIS HIPPI API to either use HIPPI-FP encapsulation or access the HIPPI physical layer directly (bypass the HIPPI-FP 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 module.

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 and the universal LAN MAC address (ULA) for each destination from a lookup table initialized at startup time from */usr/etc/hippi.imap* (for static address resolution) or built dynamically and provided by address resolution server. It obtains the local system's ULA (that is, the source ULA) from the hardware subsystem (when supported by that subsystem) or from the same lookup table (when the ULA is not provided by the hardware subsystem). It obtains the I-field value before programming the IRIS HIPPI driver to make a connection request. The lookup table maps IP addresses (or host names) to 32-bit values that are used as I-fields and to 48-bit values that are used as ULAs. The software uses each value exactly as read from the table, as summarized in Table 1-10. 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 1-10 HIPPI_LE I-Field Bit Usage

Field	Recommended Value 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 zero is required for HARP.
PS	any setting	A site may use any of the addressing formats. Logical routing is required for HARP.
C	1	1= camp-on; 0=do not camp-on. Makes HIPPI more efficient; 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 1-26. This packet conforms with the HIPPI-FP standard and the RFC 1374 guidelines.

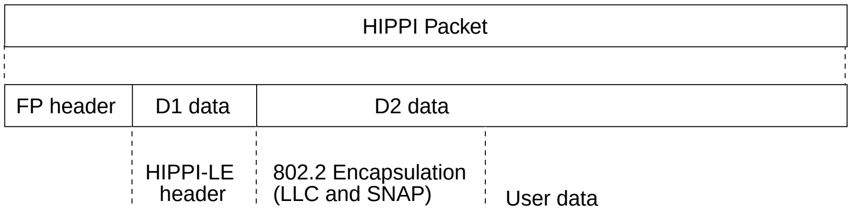


Figure 1-26 HIPPI Packet Created by IRIS HIPPI-LE

The IRIS HIPPI-LE module creates the HIPPI-FP header with the values summarized in Table 1-11.

Table 1-11 HIPPI-FP Header Creation

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 1-12. The HIPPI-LE header becomes the D1 data set for the HIPPI packet.

Note: The IRIS HIPPI-module assumes logical addressing; the address type fields are always set to logical addressing, regardless of the values obtained from the lookup table.

Table 1-12 HIPPI-LE Header Creation (D1 Data Set)

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	Least significant 24 bits of I-field, or zero	Uses the least significant 24 bits from the destination's I-field entry in the lookup table (as loaded from the *hippi.imap* file or HARP).
Destination Address Type	4 bits	0010 = logical routing	
Source Address Type	4 bits	0010 = logical routing	
Source Switch Address	24 bits	Least significant 24 bits of I-field, or zero	Uses the least significant 24 bits from the system's own I-field entry in the *hippi.imap* file, or HARP. If the file does not have an I-field entry for the source, the software uses 0x0.
Reserved	16 bits	0	
Destination IEEE Address	48 bits	48-bit ULA	Uses 48-bit address from destination's ULA entry in the lookup table (as loaded from the *hip.imap* file, or HARP), or if missing, uses 0x0.
LE Locally Administered	16 bits	0	
Source IEEE Address	48 bits	48-bit ULA	Uses the first of the following that is available: * address as provided by the hardware subsystem, or * value from the local system's own ULA entry in the lookup table (as loaded from the *hip.imap* file or HARP), or * uses 0x0

The IRIS HIPPI-LE module creates the 802.2 headers (LLC and SNAP) with the values summarized in Table 1-13. This information occupies the initial bytes of the D2 data within the HIPPI packet.

Table 1-13 802-2 Header Creation

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 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. 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.

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 HIPPI driver handles HIPPI-FP headers as summarized in Table 1-14.

Table 1-14 HIPPI-FP Header Handling

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 1-15.

Table 1-15 HIPPI-LE Header Summary

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	

Table 1-15 (continued) HIPPI-LE Header Summary

Field	Values Accepted Without Generating an Error	Comments
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 1-16.

Table 1-16 802.2 Header Bits Summary

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.

Chapter 2

Configuring IRIS HIPPI

This chapter provides instructions for and information about configuring the IRIS HIPPI software and firmware. The section “Overview of Configuration Steps” lists the configuration tasks in the order they should be performed; subsequent sections describe each task in detail. Table 2-1, at the end of this chapter, summarizes all the configurable items and indicates where the instructions for each item are located.

Overview of Configuration Steps

Before configuring the IRIS HIPPI software, decide whether or not you want the IRIS HIPPI driver to include support for the IP network stacks. The configuration steps are slightly different depending on this decision. Then, follow the appropriate set of instructions: “IRIS HIPPI Without IP Support” on page 51 or “IRIS HIPPI With IP Support” on page 52

IRIS HIPPI Without IP Support

The following steps configure the 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* or the Toolchest’s System > Software Manager 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 online document *IRIX Admin: Software Installation and Licensing*, which came with the system.
2. Optional:
   Edit the */var/sysgen/master.d/hippi* or */var/sysgen/master.d/hps* file to change the default configurations for hardware, as explained in “/var/sysgen/master.d/hippi File” on page 55 or in “/var/sysgen/master.d/hps File” on page 56.
3. Optional:
   Edit the */var/sysgen/system/hippi.sm* or *hippi_s.sm* file to exclude the IP interface and dynamic address resolution, as in “/var/sysgen/system/*.sm Files” on page 53.

Note: If you exclude IP or HARP support from the driver, and later you want to use IP or HARP, you will need to redo this configuration step and reboot the system. If you leave the IP or HARP module included, you need to only do the additional IP or HARP configuration steps to add IP or HARP functionality later. Be aware that when the driver is built to support IP or HARP, but IP or HARP is not configured, some error messages will be displayed each time the system is started.

4. The system is ready to have its IRIS HIPPI hardware installed. When it restarts (after the hardware installation), it asks you to authorize rebuilding 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 or Reconfigured Driver Into the Operating System" on page 65.

IRIS HIPPI With IP Support

The following steps describe how to 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* or the Toolchest's System > Software Manager to install the IRIS HIPPI software from the CD-ROM, as explained in the *IRIS HIPPI Release Notes*. The *inst* command is documented in *IRIX Admin: Software Installation and Licensing,* which came with the system.
2. Enable IP (that is, write **ON** into the */etc/config/network* file), as explained in "Enable IP Networking" on page 63.
3. Configure address resolution as described in "/usr/etc/ifhip.conf File" on page 56. Users that need to use static addressing must edit the */usr/etc/hippi.imap* file, as explained in "/usr/etc/hippi.imap File" on page 59.
4. Configure switch addresses to map to the logical addresses in the */usr/etc/ifhip.conf* file, as described in "Mapping ifhip.conf Addresses with Switch Ports" on page 57.
5. Edit the IP configuration files (*/etc/hosts* and */etc/config/netif.options*) to include IP network connection names and addresses, as explained in "/etc/config/netif.options File" on page 63 and "/etc/hosts File" on page 64.
6. Optional:
   Edit the */var/sysgen/master.d/if_hip* file to configure IRIS HIPPI driver parameters, as explained in "/var/sysgen/master.d/if_hip File" on page 55.

7. Optional:
   Edit the */var/sysgen/master.d/harp* file to configure IRIX HIPPI HARP parameters, as explained in "/var/sysgen/master.d/harp File" on page 55.
8. Optional:
   Edit the */var/sysgen/master.d/hippi* or */var/sysgen/master.d/hps* file to change the default configurations for hardware, as explained in "/var/sysgen/master.d/hippi File" on page 55 or in "/var/sysgen/master.d/hps File" on page 56.
9. The system is ready to have its IRIS HIPPI hardware installed. When it restarts (after the hardware installation), it asks you to authorize rebuilding 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 or Reconfigured Driver Into the Operating System" on page 65.

Note: The section "How HIPPI Ports Are Assigned to IP Interfaces" on page 39 describes how the physical HIPPI boards (*hippi0*, *hippi1*, *hippi2*, and *hippi3*) are matched to IP-over-HIPPI network interfaces (*hip0*, *hip1*, *hip2*, and *hip3*).

Checking If IRIS HIPPI Software Has Been Installed

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

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

/var/sysgen/system/*.sm Files

The files */var/sysgen/system/hippi.sm* or */var/sysgen/system/hippi_s.sm* tell the system's software which IRIS HIPPI modules to include when building the IRIS HIPPI driver into the operating system. The default file builds IP networking, dynamic address resolution (HARP), and (when available) BYPASS functionality into the driver.

If you exclude IP, HARP, or BYPASS functionality, then decide later that you want to include the functionality, you must undo your edit(s), then rebuild (with *autoconfig*) the operating system.

Edit one of the following files, depending on whether the board uses copper or fiber:

- *hippi.sm* when a copper-based IRIS HIPPI board for Challenge or Onyx is installed
- *hippi_s.sm* when a fiber-based HIPPI-Serial board for Origin or Onyx2 is installed

Caution: Do not delete any entries in this file. The operating system (kernel) cannot link properly unless each entry exists, either as an INCLUDE or EXCLUDE.

Including or Excluding IP Support

The following line in either the *hippi.sm* or *hippi_s.sm* 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
  ```

Note: When the driver is built with IP support, but the IP protocol stack is not enabled, some error messages will display each time the system is started; HIPPI still functions.

Including or Excluding HARP Functionality

The following line in the *hippi_s.sm* file can be edited to build an IRIS HIPPI driver that does not support HARP functionality:

- Original line that builds HARP support into the driver:

  ```
  INCLUDE:  harp8
  ```

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

  ```
  EXCLUDE:  harp8
  ```

Including or Excluding Bypass Functionality

The following line in the *hippi_s.sm* file can be edited to build an IRIS HIPPI driver that does not support BYPASS functionality:

- Original line that builds BYPASS support into the driver:

  ```
  INCLUDE:  hippibp
  ```

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

  ```
  EXCLUDE:  hippibp
  ```

/var/sysgen/master.d/hippi File

The */var/sysgen/master.d/hippi* file configures the IRIS parallel HIPPI hardware. Hardware configuration is optional, because all parameters have defaults. Settings in this file affect all IRIS HIPPI boards installed in the system. The IRIS HIPPI board has few configurable parameters. Specific items vary from product to product, and are explained in the file.

/var/sysgen/master.d/if_hip File

The */var/sysgen/master.d/if_hip* file configures the IRIS HIPPI IP driver. This configuration is optional, because all parameters have defaults. Settings in this file affect all IRIS HIPPI boards installed in the system. There are few configurable parameters (for example, the size for the maximum transmission unit and onboard IP checksumming). Specific items vary from release to release, so they are explained fully within the file.

/var/sysgen/master.d/harp File

The */var/sysgen/master.d/harp8* file configures IRIS HIPPI HARP parameters. Configuration is optional, because all parameters have defaults. The settings in this file affect all IRIS HIPPI boards installed in the system. There are few configurable parameters. The specific items vary from release to release, so they are explained fully within the file.

/var/sysgen/master.d/hippibp File

The */var/sysgen/master.d/hippibp* file configures the IRIS HIPPI bypass driver. This configuration is optional, because all parameters have defaults. Settings in this file affect all IRIS HIPPI boards installed in the system. There are few configurable parameters. The specific items vary from release to release, so they are explained fully within the file.

/var/sysgen/master.d/hps File

The */var/sysgen/master.d/hps* file configures the IRIS HIPPI serial driver. Configuration is optional, because all parameters have defaults. The settings in this file affect all HIPPI boards installed in the system. Configurable parameters are explained within the file.

/usr/etc/ifhip.conf File

This section describes how network addresses are dynamically mapped (resolved) to physical addresses in a HIPPI fabric. This section assumes that the reader is familiar with the standard Internet ARP (RFC 826, *Ethernet Address Resolution Protocol*) and Inverse ARP (RFC 1293, *Inverse Address Resolution Protocol*).

In the */usr/etc/ifhip.conf* file, you specify the HARP server and the local host address. This file is read by ifhipconfig(1m) during system startup. It is run (called by */etc/init.d/hippi*) before the network start-up script (*/etc/init.d/network*) so that these assignments are ready before ifconfig(1m) brings up the logical interfaces.

Using HARP is optional. The default *ifhip.conf* configuration file does not specify a server, and HARP is not used. In this case all HIPPI hardware addresses must be specified in the */usr/etc/hippi.imap* file.

Guidelines for Selecting a HARP Server

Select hosts within each logical IP subnet (LIS) to perform the HARP server function. The following guidelines should be followed in selecting this system:

- The system and its HIPPI fabric connection must be available (operational) whenever any member of the LIS wants to access another IP host over HIPPI.
- The system should not be used for lengthy or infinite data transfers over its HIPPI fabric connection. The link to the HARP server must become available frequently so that client requests can get through in a timely manner. In most cases, the amount of traffic on the system’s HIPPI link does not affect performance; large numbers of small-sized datagrams or data movements do not present a problem, whereas even one lengthy transfer could negatively affect many members of the LIS.
- The HARP server has only a small overhead, little more than the clients, so this is not a major consideration in choosing the server.

ifhip.conf File Syntax

Here is an example of a *ifhip.conf* file; words in bold must be typed as they appear:

```
hip0 arpserver 0x07000FE0 FF:FF:FF:FF:FF:FF myhipaddr 0x07000006
```

In this example, `0x07000FE0` is the recommended address of the HARP server. The actual address of the host must be entered in the *ifhip.conf* file. In the example above, the host is a client connected to the switch at a port configured as logical address 6.

Here is an example of a HARP server's *ifhip.conf* file with the default HARP server address as well as the ULA hard-coded into *ifhip.conf*:

```
hip0 arpserver 0x07000FE0 08:00:63:41:42:43 myhipaddr 0x07000FE0
```

In this case the ULA of the server was previously determined with the *hipcntl getmac* command. In any case the ULA is an optional field.

There must be one line (entry) for each IRIS HIPPI port that supports IP. The HIPPI-LE module obtains this information during startup from the *ifhip.conf* file; it obtains this information only during startup and stores it. Subsequent changes to and reloading of the *ifhip.conf* file (for example, *ifhipconfig new_entry*) do not affect the HIPPI-LE module's stored values. To change these values, you must edit the file and restart networking with */usr/etc/network stop* and */usr/etc/network start*, configure the interface(s) down and up again with */usr/etc/ifconfig*, or reboot the system.

Note: If you assign a hardware address (ULA or MAC) to an IRIS HIPPI port, it is highly recommended that you use the value 8 for the most significant byte of the address, for example, 08:*xx*:*xx*:*xx*:*xx*:*xx*.

Mapping ifhip.conf Addresses with Switch Ports

Once you have decided on your server (and client) addresses, be sure to configure the switch ports appropriately. For example, if you are using the recommended HARP server address of 0x07000FE0, and the host is connected to switch port 00, you should configure the switch port 00 to map to logical address 0x07000FE0. If you are not using the recommended HARP server address of 0x07000FE0, configure the HARP server switch port to the correct default logical address which you have specified in */usr/etc/ifhip.conf*.

Using Multiple HARP Servers

You can configure more than one host to be the HARP server in an LIS to give added reliability. If one HARP server goes down or becomes unavailable for any reason, the clients will automatically use another HARP server. When the server again becomes available, the clients will recognize its return and use it as necessary.[1]

To specify multiple HARP servers, add an additional line of logical address and ULA for the backup HARP server. For example, here is the contents of a *ifhip.conf* file for an LIS with backup HARP server:

```
hip0 arpserver 0x07000FE0 08:00:69:04:62:a7 arpserver 0x07000001 08:00:69:04:79:86
hip0 myhipaddr 0x07000007
```

Figure 2-1 illustrates a part of an LIS in which two HARP servers, a client, and switch are configured.

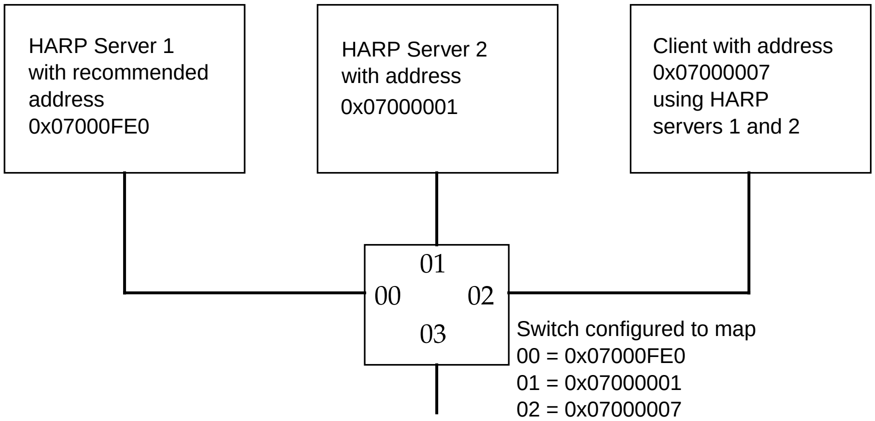


Figure 2-1 Example of Multiple HARP Servers and Switch Port Configuration

[1] Multiple HARP servers in conjunction with Silicon Graphics optional IRIS FailSafe product allow you to construct a high-availability network.

/usr/etc/hippi.imap File

You can configure a *hippi.imap* file if you have an endpoint within the HIPPI fabric that does not dynamically configure an address at system startup. These might be systems with older software, for example. The entries can be placed into the *hippi.imap* file that the HARP software automatically loads into the kernel-resident database when it starts, or the entry can be added directly to the database by using the *hipmap* command.

The */usr/etc/hippi.imap* file maps network connection names (or "inet" addresses) to HIPPI I-fields and universal LAN MAC (ULA) addresses.[1] It contains an entry for each remote (destination) endpoint. For endpoints on which the HIPPI hardware that does not store a hardware address[2], a 6-byte IEEE address (called a ULA or MAC address) can also be mapped. The file can contain up to 1301 lines (mappings).

Note: The I-field can be set to any value supported by each system's implementation; both endpoints can even use the same value. IRIS HIPPI supports all values, including 0x00000000.

To load this file into memory or change the loaded table's contents, follow instructions in "Change the Static ARP Table That Maps IP Addresses to I-fields" on page 77.

Each time the HIPPI-LE module is about to program the HIPPI subsystem to issue a connection request, it obtains values for the I-field and ULA of its destination from the HARP lookup table. The lookup table maps IP addresses or network connection names to HIPPI addresses. This table is generated at startup time when clients register (see "HARP Registration Phase" on page 19) and, optionally, from the *hippi.imap* file. The HARP table can be modified in real time with the *hipmap* command. HIPPI clients usually obtain mappings dynamically by means of the HARP protocol (see "HARP Operation Phase" on page 20).

[1] Network connection names are sometimes, inaccurately, called hostnames. For example, a system with a hostname of *goofy* might have network connection names of *hippi1-goofy*, *hippi2-goofy*, and *gateway-goofy*.

[2] The IRIS HIPPI boards for the CHALLENGE and Onyx platforms do not store ULA (MAC) addresses.

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

- *name* `0x`*XXXXXXXX*

  name is the network connection name 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 address format for port1
  ```

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

  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 address format
  ```

Each ULA is a 48-bit value. To assign a ULA, the line (entry) in the file must have the format illustrated below:

- *name* `0x`*XXXXXXXX* *XX*:*XX*:*XX*:*XX*:*XX*:*XX*

  name is the network connection name or the IP address as described in the two examples above, `0x`*XXXXXXXX* is the 32-bit I-field in hexadecimal notation, and *XX*:*XX*:*XX*:*XX*:*XX*:*XX* is the 48-bit ULA (MAC address) in hexadecimal notation. The following lines are examples:

  ```
  hippi-goofy  0x01000001  05:A6:70:9B:FF:8E
  223.9.1.241  0x01000001  05:A6:70:9B:FF:8E
  ```

I-Field Templates

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, if logical addressing is used, the source’s address.

Figure 2-2 is a template that can be used for developing each 32-bit I-field for a logical addressed environment; Figure 2-3 is for developing source addressed I-fields. The templates show the values that Silicon Graphics recommends for use by the IRIS HIPPI-LE upper layer protocol.

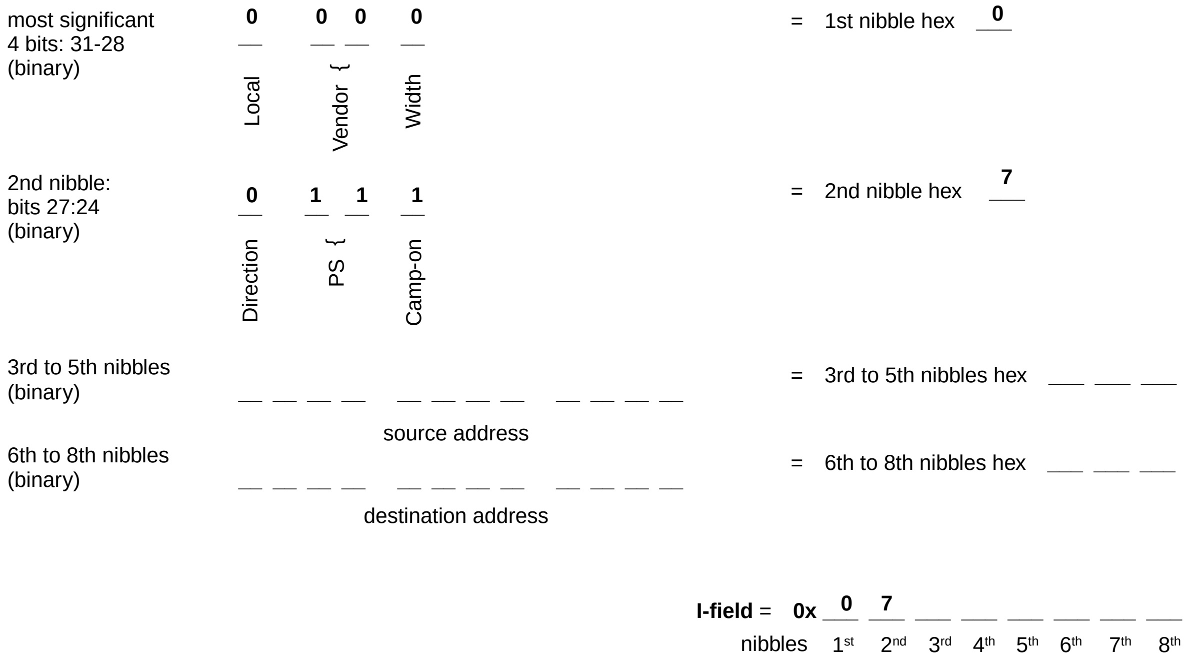


Figure 2-2 Template for Creating Logical Address I-Fields With Recommended Values

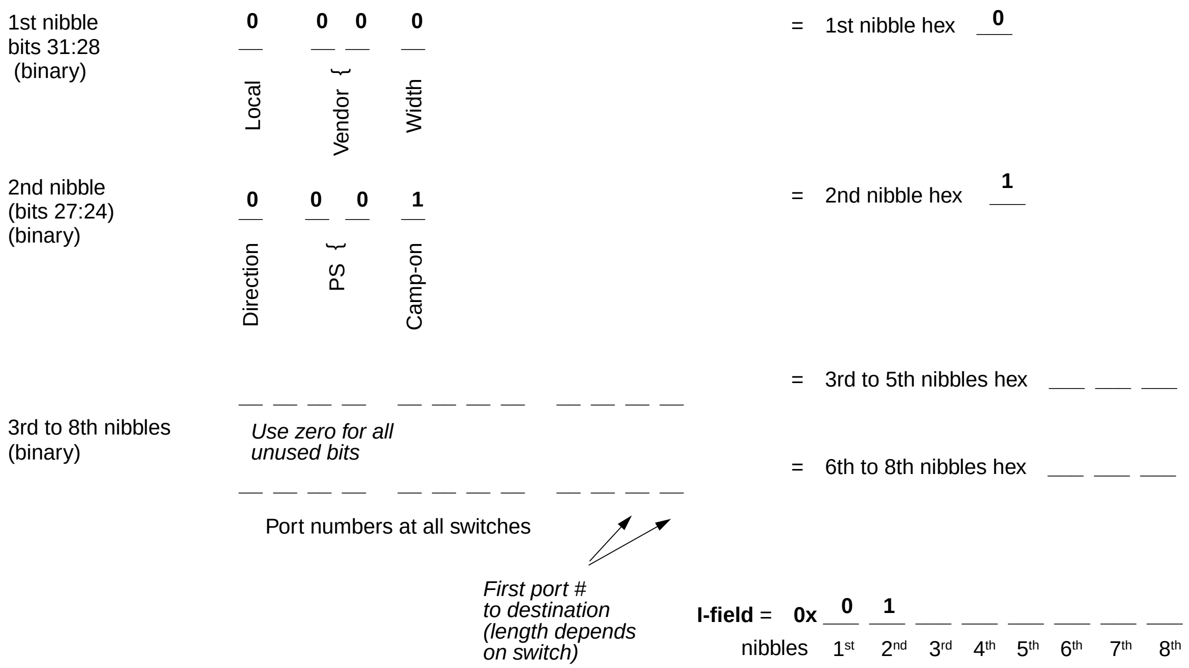


Figure 2-3 Template for Creating Source Address I-Fields With Recommended Values

Editing IP Configuration Files

To configure the IP networking interfaces, edit the files */etc/config/network*, */etc/config/netif.options*, */etc/hosts*, and */etc/config/ifconfig-#.options* files, as described in this section.

The *network* file starts the IP software. Each time it starts, it uses information from the *hosts* and *netif.options* files to configure the IP interfaces.

Note: For additional details about enabling the IP networking software and configuring network interfaces, refer to the *IRIX Admin: Networking and Mail*, which is available online through IRIS InSight or in hardcopy.

Enable IP Networking

To have the IP network software automatically start operating 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 or use the command *chkconfig network on*.

Note: Enabling IP networking does not result in IP over HIPPI; it only enables the IP software within the operating system to operate over whatever drivers are available to service it. The other IP configuration files associate the IP stack with specific drivers.

/etc/config/netif.options File

The */etc/config/netif.options* file maps local network connection names (or IP addresses) to IRIS HIPPI network interfaces (for example, *hip0*, *hip1*, and so on.). There must be a two-line entry for each IRIS HIPPI port 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 name or IP address in this file must also be in the */etc/hosts* file; the name or IP address in the *netif.options* must match exactly the name or IP address in the *hosts* file.

Systems that function as a client or server for *bootp* should configure Ethernet as their primary network interface. HIPPI cannot be the primary interface. 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 network interfaces, an FDDI interface (that is, *xpi0*), and a primary Ethernet interface. If this system's hostname is *goofy*, these IRIS HIPPI entries work with the examples of */etc/hosts* file entries in the next section.

```
if1name=et0
if1addr=$HOSTNAME

if2name=xpi0
if2addr=fddi-$HOSTNAME

if3name=hip0
if3addr=hippi1-$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.

/etc/hosts File

The */etc/hosts* file maps names to network-layer IP addresses. There must be one entry for each IRIS HIPPI port that is to support IP traffic. The entries must be similar to the examples below, which illustrate four IRIS HIPPI interfaces for a system whose hostname is *goofy*:

```
223.9.1.2     hippi1-goofy.toons.com   hippi1-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
```

Note: A local */etc/hosts* file must exist and must contain the local IRIS HIPPI network interfaces, even when the site uses NIS or DNS.

/etc/config/ifconfig-#.options File

Each */etc/config/ifconfig-#.options* file configures one IP interface. The # in the filename matches the numeral in the `if#name` entry in the *netif.options* file. Each instance of this file is optional; however, to obtain optimal performance on an IRIS HIPPI interface, the size of the windows (that is, the amount of outstanding and in_transit data) for sending and receiving TCP/IP datagrams) must match the window size used on remote systems. If possible, the IRIS HIPPI default value (524288 bytes) should be configured on all the HIPPI interfaces within the HIPPI fabric. If this default cannot be used throughout the fabric, you must create a file named *ifconfig-#.options* for the IRIS HIPPI interface and set the TCP windows value to the value selected for the HIPPI fabric.

To configure the TCP window size (in bytes), create a file named *ifconfig-#.options* for the IRIS HIPPI interface and place these lines in the file:

```
sspace nnnnnn
rspace nnnnnn
```

nnnnnn is any value, divisible by 1024, between 1024 and 524288.

Note: For large-sized TCP windows to work, the *tcp_winscale* and *tcp_tsecho* variables in the */var/sysgen/master.d/bsd* file must each be set to 1 (their defaults).

If the memory used by TCP/IP applications is an issue, you can obtain nearly full performance by using 262144 (256 * 1024), instead of the default 524288 (512 * 1024).

For a complete description of the IP parameters that can be configured in this file, see the ifconfig(1M) man page.

Building a New or Reconfigured Driver Into the Operating System

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.

For the IRIS HIPPI subsystem to be functional, the IRIX operating system (kernel) that is currently on the system must be rebuilt to include the IRIS HIPPI driver. When changes are made to any of the following files, or when new IRIS HIPPI software is installed, it is necessary to rebuild the operating system:

- */var/sysgen/ioconfig/hippi*
- */var/sysgen/master.d/harp*
- */var/sysgen/master.d/hippi*
- */var/sysgen/master.d/hippibp*
- */var/sysgen/master.d/hps*
- */var/sysgen/master.d/if_hip*
- */var/sysgen/system/hippi.sm*
- */var/sysgen/system/hippi_s.sm*

The three sets of instructions below each build a new operating system and start it running. It is not important which set of instructions is 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 name
configuring hip1 as name
```

Instruction Set 2

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

When the system shuts down, restart it. When the question about automatically reconfiguring the operating system 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:

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

List of All Configurable Items

Table 2-1 lists all IRIS HIPPI's configurable items, and summarizes the following for each item: whether configuration for the item is optional or required, what the default setting is, and where the configuration instructions for each item reside.

Table 2-1 Alphabetic Summary of HIPPI Configuration Items

Item	Configuration Required/Optional	Default Setting	Location of Instructions
Bypass module (build it into or exclude it from IRIS HIPPI driver; module is not built into driver by default)	O	Not included	page 53
Driver	O	See MTU and TCP/UDP checksum	page 53 and page 55
HARP module (build it into or exclude it from IRIS HIPPI driver; by default it is included)	O	Included	page 55
Hardware	O	See file	page 55
Address resolution mechanism:			
arpserver	O	None	page 56
myhipaddr	O	None	page 56
I-field for a HIPPI port:			
local port (source)	R (HARP only)		
remote ports (destinations)	R (static ARP only)	None exists	page 56

Table 2-1 (continued) Alphabetic Summary of HIPPI Configuration Items

Item	Configuration Required/Optional	Default Setting	Location of Instructions
ULA for a HIPPI port:			
local port (source)	O	No ULA	page 56
remote ports (destination)	O	None exists	page 59
IP address to IP network connection name mappings	R for IP traffic	None exists	page 64
IP module (build it into or exclude it from IRIS HIPPI driver; by default it is included)	O	Included	page 53
IP network interface	R for IP traffic	None exists	page 63
IP networking: enable/disable	R for IP traffic	Disabled	page 53 and page 63
IP-over-HIPPI: list of configuration steps	R for IP traffic	None exists	page 52
Maximum transmission unit (MTU) size	O	65280	page 55
TCP + UDP checksums in hardware	O	Enabled	page 55
TCP window size	O	524288 bytes	page 65

Chapter 3

Maintaining, Monitoring, Verifying, and Troubleshooting IRIS HIPPI

This chapter tells how to maintain, monitor, and troubleshoot the IRIS HIPPI subsystem.

Commands Available for IRIS HIPPI

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

Table 3-1 HIPPI Command Summary

Command	Purpose	Page
/usr/etc/hipmap	Displays, adds, or deletes entries from the in-memory HARP table, which maps HIPPI hardware addresses to IP addresses.	page 77
/usr/etc/hipcntl	Provides a variety of control and status functions for the IRIS HIPPI subsystem.	page 70, page 71, page 78
/usr/etc/hiptest	Verifies IRIS HIPPI subsystem through the character device interface, without going through the IP network interface.	page 83
/usr/etc/ping	Verifies IP network interfaces. Can be used to verify that a HIPPI network interface is functioning.	page 87,
/usr/etc/ifhipconfig	Configures parameters of the HIPPI dynamic ARP (HARP) interface such as arpserver and myhipaddr.	page 76
/usr/etc/ifconfig	All the normal IP configuration options work with IRIS HIPPI IP network interfaces (that is, *hip#*), arp, and the specification of a destination IP address for setting up a point-to-point connection.	page 76
/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.	page 87, page 90

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 HIPPI utilities (*hipmap*, *hipcntl*, *hiptest*, and *ifhipconfig*) 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 command resets the board; all data (incoming or outgoing) that is on the board is lost. This event leaves the board in a non-operational state: incoming connection requests are rejected and applications attempting to transmit or receive over the device receive ENODEV errors.

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

To start or enable the IRIS HIPPI board (after a shutdown), 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 before starting the firmware.

```
# 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 and its source (SRC) and destination (DST) subsystems, use the command below. Most of the counted items are initialized to zero upon reset of the board and roll over to zero upon reaching 2^{32} (that is, at 4,294,967,295); exceptions are explained in the tables. The displayed information is described in Table 3-2 or Table 3-3, depending on the system's hardware. Troubleshooting advice regarding this information is provided in "Interpreting Status Information" on page 92.

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

Table 3-2 hipcntl Display Fields (Challenge and Onyx)

Status Item	Description
FLAGS:	
DSIC	SRC sees the incoming **INTERCONNECT** signal from its destination.
SDIC	DST sees the incoming **INTERCONNECT** signal from its source.
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 asserted
SRC connections:	Count of total connection **REQUEST** signals issued by source.
SRC packets:	Count of total packets sent by source.
SRC rejects:	Count of SRC's connection attempts that were rejected by the remote destination.
SRC seq errors (dm):	Count of sequence errors detected within the SRC's data state machine.
SRC seq errors (cd):	Count of illegal sequencing of inbound control signals at SRC's connection state machine. The remote destination is believed to be at fault.

Table 3-2 (continued) hipcntl Display Fields (Challenge and Onyx)

Status Item	Description
SRC seq errors (cs):	Count of sequence errors detected within the SRC’s connection state machine.
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 SRC withdrew the request.
SRC connects lost:	Count of connections that were dropped by the remote destination.
SRC parity errs:	Count of SRC’s parity errors.This error indicates that a local parity error (for example, on the IRIS HIPPI board) resulted in the transmission of invalid data.
DST connections:	Count of connections accepted by DST.
DST packets:	Count of total packets received
DST rcv on bad ulp:	Count of packets discarded by DST due to unknown ULP-id.
DST hippi-le drop:	Count of HIPPI-LE packets discarded by DST.
DST llrc:	Count of connections dropped by DST due to LLRC errors.
DST parity:	Count of connections dropped by DST due to parity errors in incoming data. Bit-error rates below 10^{-20} are not counted; for example, if the DST encounters 1 error after receiving 10^{-21} bits of correct data, it does not drop the connection.
DST sequence err:	Count of connections dropped by DST due to sequence errors. Sequence errors are invalid combinations of HIPPI-PH signals (for example, an asserted **BURST** signal when the **PACKET** signal is not asserted).
DST sync err:	Count of synchronization errors detected by DST.
DST illegal burst:	Count of inbound packets of an illegal burst size.
DST sdic lost:	Count of connections dropped by DST due to lost **SDIC** signal.
DST null connections	Count of connections with zero-length packets.

Table 3-3 hipcntl Display Fields (Origin and Onyx2)

Status Item	Description
FLAGS:	
LOOPBACK	The board is operating in internal loopback mode. Packets are not being passed to the fiber; instead they are going directly from the board's SRC to its DST. When this flag is absent (not set), the board is operating in normal mode; packets are transmitted onto the fiber.
DST.SIG_DET	HIPPI-Serial DST is detecting a signal on inbound fiber. Normal operation requires the presence of this flag.
DST.LNK_RDY	HIPPI-Serial DST is in operational state (that is, it has successfully completed reset and is currently in HIPPI-Serial link state 2). This flag is absent (not set) only if the DST state machine transitions to state 0 or 1, caused by losing contact with the HIPPI-Serial (G-link) chip on the board or the fiber connection. Normal operation requires the presence of this flag.
DST.FSYNC	HIPPI-Serial DST is seeing/interpreting the Flag (alternating 0/1) bit from the coding nibbles of the incoming data frames. Normal operation requires the presence of this flag.
DST.OH8SYNC	HIPPI-Serial DST is synchronizing with the OH8 framing (alternating 0/1) overhead bit. Normal operation requires the presence of this flag.
ACCEPTING	DST is accepting connections. When this flag is not listed, the DST is rejecting connections. Normal operation requires the presence of this flag.
DST.PKT	DST sees that the inbound **PACKET** signal is asserted.
DST.REQ	DST sees that the inbound **REQUEST** signal is asserted.
SRC.REQ	SRC channel's outbound **REQUEST** signal is asserted.
SRC.CON	SRC sees that the inbound **CONNECT** signal is asserted.
SRC connections:	Count of connection **REQUEST** signals issued by source.
SRC packets:	Count of total packets sent by local SRC.
SRC rejects:	Count of SRC's connection attempts that were rejected by the remote destination.

Table 3-3 (continued) hipcntl Display Fields (Origin and Onyx2)

Status Item	**Description**
SRC xmit retry:	Count of attempts driver made to retransmit packets that were interrupted, either because the remote destination disconnected, or because there was a parity error while transmitting the packet.
SRC glink reset:	Count of times that the firmware reset both the HIPPI-Serial (G-link) chips. A reset occurs when the firmware believes one of the HIPPI-Serial (G-link) chips is not responding. However, a reset can be caused by faulty cabling, ODLs, or connectors since the firmware cannot identify the true cause for an unresponsive HIPPI-Serial portion of the board.
SRC glink lost;	Count of times that the firmware fails to see any one of the following flags for more than half a second: DST.OH8SYNC, DST.FSYNC, DST.LNK.RDY, or DST.SIG.DET. This event can be counted, at maximum, 50 times per second (at 25MHz).
SRC time outs:	Count of connection attempts by SRC that timed out.
SRC connects lost:	Count of connections made by SRC that were dropped by the other side.
SRC parity errs:	Count of SRC parity errors. This error indicates that a local parity error (for example, on the IRIS HIPPI board) resulted in the transmission of invalid data.
SRC number bytes sent:	Count of bytes transmitted. Maximum count, before starting over at zero, is 2^{64} (that is, more than 18 quintillion).
DST connections:	Count of connections that were accepted by DST.
DST packets:	Count of total packets received by DST.
DST rcv on bad ulp:	Count of HIPPI-FP packets that DST discarded due to unknown ULP-id. Some programs produce a few of these events when they are terminated unexpectedly (for example, doing a **Ctrl-C** to *hiptest*), because the receiver goes away before the transmitter.
DST hippi-le drop:	Count of HIPPI-LE packets discarded by DST due to no accessible space on the receive packet queue.
DST llrc:	Count of bursts received by DST with LLRC errors.

Table 3-3 (continued) hipcntl Display Fields (Origin and Onyx2)

Status Item	**Description**
DST parity:	Count of bursts received by DST with parity errors. Bit-error rates below 10^{-20} are not counted; for example, if the DST encounters 1 error after receiving 10^{-21} bits of correct data, it does not drop the connection.
DST frame/state err:	Count of (1) framing (alternating OH8 overhead bit) errors that occurred while **PACKET** signal was asserted, and (2) illegal HIPPI signal states or transitions (for example, a **PACKET** signal was received without a preceding **REQUEST** signal).
DST flag err:	Count of data frame alternating flag bit synchronizations that were lost while **PACKET** signal was asserted.
DST illegal burst:	Count of packets with illegal burst sizes.
DST link rdy lost in pkt:	Count of packets that were aborted due to the DST.FSYNC, DST.OH8SYNC, or DST.LNK.RDY flag becoming unset (not true) when the **PACKET** signal was asserted.
DST null connections:	Count of connections over which no data was sent.
DST ready errors:	Number of bursts received for which no **READY**s had been sent.
DST bad packet starts:	Count of inbound HIPPI packets that encountered errors almost immediately (for example, a **PACKET**-**BURST**-no_**PACKET** sequence occurred for a HIPPI-FP connection but less than 12 bytes was transmitted, or a **PACKET** signal was asserted then deasserted without a **BURST** ever occurring).
DST number bytes received	Count of bytes received. Maximum count, before starting over at zero, is 2^{64} (that is, more than 18 quintillion).

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
```

Change IP Network Interface Parameters

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

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

Note: Some of the standard *ifconfig* arguments are not supported for IRIS HIPPI (for example, broadcast and arp).

Configuring HIPPI Dynamic ARP (HARP)

The *ifhipconfig* command configures HIPPI logical network interfaces for HARP.

To display the HIPPI parameters for, say, hip0:

```
# ifhipconfig hip0
```

To configure interface hip0 to a HIPPI hardware address of 7:

```
# ifhipconfig hip0 myhipaddr 0x07000007
```

To set a hip1 HARP server address:

```
# ifhipconfig hip1 arpserver 0x07000FE0
```

Refer to the ifhipconfig(1M) reference page for more information.

Change the Static ARP Table That Maps IP Addresses to I-fields

The */usr/etc/hipmap* command makes changes to the address resolution information (that is, the address lookup table) that is currently in memory. Any *name* or *IPaddress* used on the *hipmap* command line must already exist in the */etc/hosts* database.

Note: Any changes made to the lookup table using the command line will be lost when the */etc/init.d/network* script is invoked. To make changes that endure, follow the instructions in "/usr/etc/hippi.imap File" on page 59.

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

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

where *name* is an entry from the */etc/hosts* database (either an IP address or its mnemonic).

To delete one entry from the lookup table, use one of these command lines:

```
# hipmap -d name
# hipmap -d IPaddress
```

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

```
# hipmap -D
```

To concatenate new entries and/or replace already existing entries with new values from a file, use this command line:

```
# hipmap -f filename
```

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

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

Display the ARP Table That Is Currently in Memory

Use the command line below to display the currently-in-memory address resolution information (that is, the lookup table of IP addresses with their mappings to I-fields and ULAs):

```
# 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.

For the HIO board, the granularity for this timeout is a quarter of a second (that is, 250 milliseconds); a command-line *timeout* value that are not divisible by 250 is rounded up to the next quarter-second. For the XIO board, the granularity is 1 millisecond. In this command line, the *timeout* is expressed in milliseconds.

```
# hipcntl [hippi#] stimeo  timeout_in_milliseconds
```

Note: It is possible to set the timeout to a value that is so small that the source closes its connections to one or more destinations before they have enough time to respond. The symptom of this is many `SRC time outs` and fewer than expected `SRC bytes sent`, as reported by *hipcntl status*.

Display ULA (MAC) Address

Use the command line below to display the hardware address:

```
# hipcntl [hippi#] getmac
```

Note: This functionality does not work for the copper-based HIO mezzanine board because that hardware does not store a hardware address. The ULA for an HIO board is configured in the *hippi.imap* file and can be displayed with the *hipmap -a* command.

Installing a Loopback Link

To install a loopback link, follow the instructions in “Loopback Link for Challenge or Onyx Systems” or “Loopback for Origin and Onyx2 Systems,” as appropriate.

Loopback Link for Challenge or Onyx Systems

To install a copper loopback link on an IRIS HIPPI board installed into a CHALLENGE or Onyx system, use any of the procedures illustrated below:

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

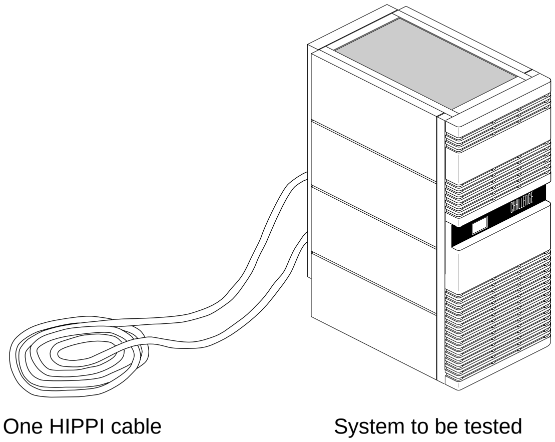


Figure 3-1 Installing a Copper Loopback Link Using a HIPPI Cable

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

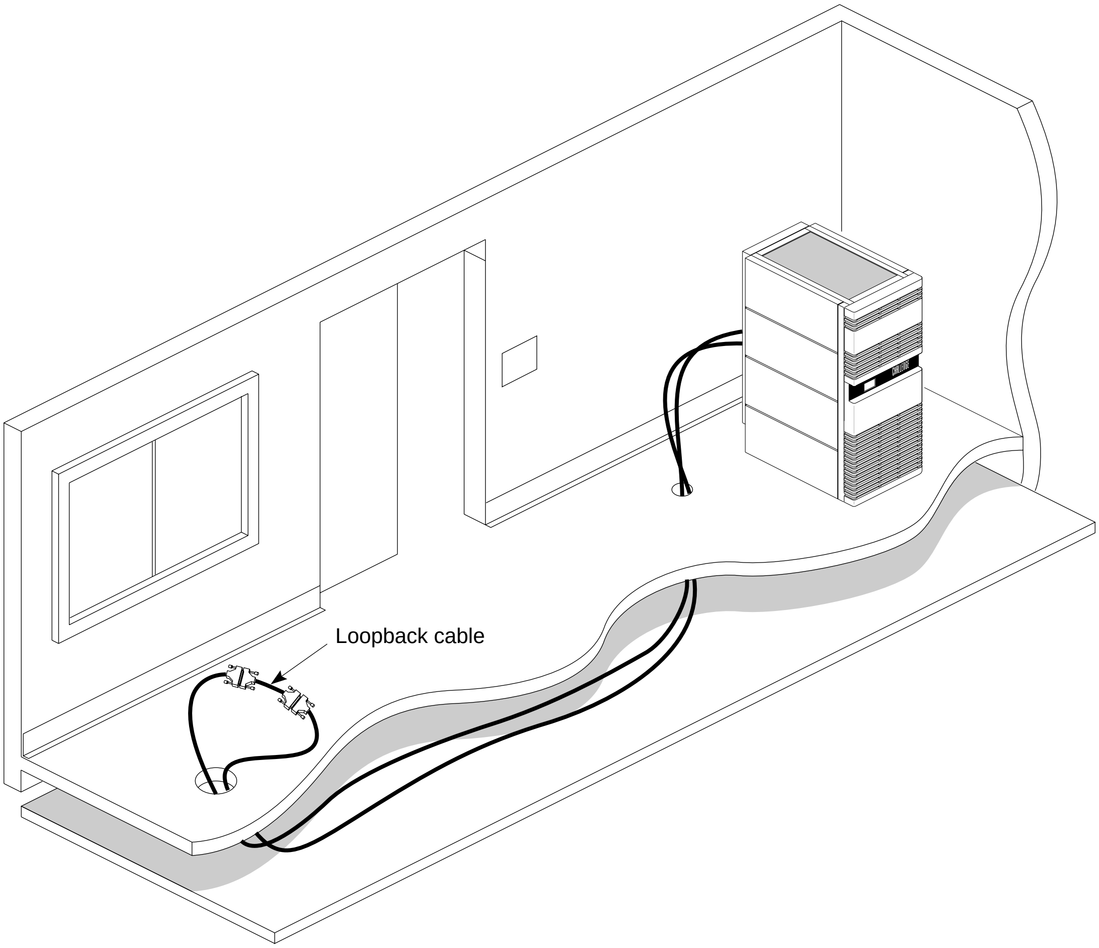


Figure 3-2 Installing a Copper Loopback Link Using a Loopback Cable

Note: An alternate method for looping back the SRC and DST is to attach both IRIS HIPPI channels (SRC and DST) to the same port on a switch, and to add a HIPPI address mapping (for this connection) to the switch's address table. The *hiptest* utility can then transmit packets to itself by using an I-field that contains the host's own HIPPI address as the destination. This works because, unlike many IRIX drivers, the IRIS HIPPI driver does not automatically route self-addressed packets through the loopback interface (*lo0*).

Loopback for Origin and Onyx2 Systems

To install a loopback link for an IRIS HIPPI-Serial XIO board installed in an Origin or Onyx2 system, use any of these procedures:

- Use the *hipcntl loopback* command (as demonstrated below) to configure the board for internal (board) loopback.

  ```
  # hipcntl hippi# shutdown
  # hipcntl hippi# loopback
  # hipcntl hippi# startup
  ```

  where # is the unit number for the board as displayed by *hinv -d hippi*.

  Note: With this type of loopback, the optical transmitter and receptor (ODLs) on the board are not verified during the verification procedures. The card runs in loopback mode until it is reset (that is, until the *hipcntl* shutdown/startup sequence is invoked).

- Attach a dual-SC, multimode loopback connector to the system's I/O panel plate. This special device, illustrated in Figure 3-3, connects the IRIS HIPPI-Serial DST (receive) and SRC (transmit) fibers to each other.
- Attach the IRIS HIPPI-Serial port to a switch and add a HIPPI address mapping for the port to the switch's address table. The *hiptest* utility can then transmit packets to itself by using an I-field that contains the host's own HIPPI address as the destination.

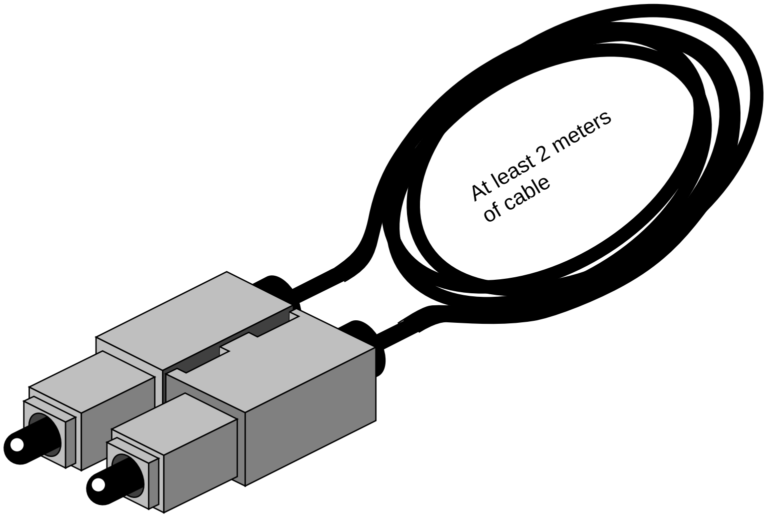


Figure 3-3 Fiber-optic Loopback Connector

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 Board and Its HIPPI-FP Interface" on page 83 in this section. After the HIPPI subsystem has been verified, further upper-layer verification and interconnectivity tests can be run (for example, the test described under the heading "Verify an IP-over-HIPPI Interface" on page 87 in this section) with the system attached to other HIPPI systems (for example, a switch or endpoint).

Note: Unlike many IRIX drivers, the IRIS HIPPI driver does not automatically route self-addressed packets through the local loopback interface (*lo0*), so that even the IP stack can be verified with the loopback link in place.

Verify That the Board Has Been Located by the Software

To verify that an IRIS HIPPI board has been located by the operating system during the last reboot, use any of the following commands:

```
% hinv -d hippi
HIPPI-Serial adapter: unit #, in module # I/O slot #
```

```
% hinv -mvv -d hippi
...
Location: /hw/module/1/slot/io6/hippi_serial
  HIPPI-SERIAL Board: barcode ######     part 030-0968-00# rev #
  Group ff Capability ffffffff Variety ff Laser 0000000a0f8f
HIPPI-Serial adapter: unit #, in module # I/O slot #
```

```
% find /hw/module -name hippi
/hw/module/#/slot/io#/hippi_serial/pci/0/hippi
```

Note: Each IRIS HIPPI board may have multiple full-path entries in the hardware graph; the `pci/0/hippi` entry is the main one.

Verify the Board and Its HIPPI-FP Interface

To verify the IRIS HIPPI board and its HIPPI-FP interface (without going through the IP stack), use the */usr/etc/hiptest* utility. This test works for an IRIS HIPPI port that has a loopback link installed between its source and destination channels (see “Installing a Loopback Link” on page 79 for instructions), or it can be invoked as sender or receiver alone, in which case 2 instances of the utility must be running to successfully exchange packets between the 2 processes. The utility requires the user to be superuser (root).

Fast and Quick Verification Test

For a simple, quick verification test through a physical port that has a loopback link installed, use the commands below:

```
% cd /usr/etc
% su
Password: password
# hinv -d hippi
<use the displayed unit number in the next command line>
<for my_ifield, use the port’s own I-field from the hippi.imap file>
# hiptest -I 0xmy_ifield -D /dev/hippi#
hiptest: /dev/hippi#:
sending 64 packets, size range [16..16777224] to I-field 0x00000001
...................... <up to 64 dots>
```

For a simple, quick verification test when a physical port is attached to a switch, use the commands below:

```
<in one UNIX shell>
% cd /usr/etc
% su
Password: password
# hinv -d hippi
<use the displayed unit number for the # in the next command line>
# hiptest -r -D /dev/hippi#
hiptest: /dev/hippi#:
reading 64 packets
...................... <up to 64 dots>
hiptest(DST): received 64
```

```
<in a different UNIX shell>
% cd /usr/etc
% su
Password: password
<use the same or a different installed device/unit number as above>
<use the port's I-field from the hippi.imap file>
# hiptest -s -I 0xmy_ifield -D /dev/hippi#
hiptest: /dev/hippi#:
sending 64 packets, size range [16..16777224] to I-field 0xmy_ifield
...................... <up to 64 dots>
hiptest(SRC): sent 64
```

Extensive Verification Test

The *hiptest* utility sends randomly-sized HIPPI-FP packets that contain randomly generated data for 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 utility 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. Uses ULP-id value 0x89 (hexadecimal).

D1 area size — 8 bytes.

D1 data set — Zero.

D2 area size — Randomly generated size, ranging from 16 up to the constraining byte count specified by *maxsize*. The absolute maximum size possible is 2MB for copper-based HIO cards or 16MB for fiber-optics XIO cards. The first words of the D2 area are included in the first burst of the packet.

D2 data set — Randomly generated data ranging.

The utility allows you to specify the following packet characteristics:

-I

The I-field value (in hexadecimal format) to use for the source's connection request. The value must exist in the *hippi.imap* file. There is no default; user must specify an I-field.

```
# /usr/etc/hiptest -I 0x07001002
hiptest: /dev/hippi0:
sending 64 packets, size range [16..2097160],
to I-field 0x07001002
```

-D

The IRIS HIPPI board (unit) to test: */dev/hippi0*, */dev/hippi1*, etcetera. The unit number can be displayed with the *hinv -d hippi* command. If the board is not specified on the command line, the command uses */dev/hippi0*.

```
# /usr/etc/hiptest -I 0x01000005 -D /dev/hippi3
hiptest: /dev/hippi3:
sending 64 packets, size range [16..16777224],
to I-field 0x01000005
```

maxsize

The maximum bytesize (in decimal format) for the packets. The value specified must be a number that is divisible by 8. The minimum is 16 (8 bytes of FP header and 8 bytes of D1 data). The maximum is 2 megabytes for a copper-based HIO board or 16 megabytes for a fiber-optics XIO board, plus 8 bytes (that is, 2097160 or 16777224 bytes). If *maxsize* is not specified on the command line, the command uses the maximum possible for the installed hardware.

```
# /usr/etc/hiptest -I 0x00000001 8192
hiptest: /dev/hippi0:
sending 64 packets, size range [16..8192],
to I-field 0x00000001
```

#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; if *#packets* is not specified on the command line, *hiptest* uses 64. When *#packets* is specified, *maxsize* must also be specified, since the first argument is always interpreted as the *maxsize* and the second as the *#packets*.

```
# /usr/etc/hiptest -I 0x01000002 8192 10
hiptest: /dev/hippi0:
sending 10 packets, size range [16..8192],
to I-field 0x01000002
```

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-N]] -r -s [maxsize [#pckts] ]
```

Examples:

- To run the test through the loopback mechanism using the default settings, use the commands below:

  ```
  % cd /usr/etc
  % su
  Password: thepassword
  # hiptest -I 0xmy_ifield
  hiptest: /dev/hippi0:
  sending 64 packets, size range [16..2097160],
  to I-field 0xmy_ifield
  ...................... <up to 64 dots>
  ```

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

  ```
  # hiptest -I 0xmy_ifield 16 1
  ```

- To send 25 packets of up to 2 megabytes through the loopback mechanism, use the command line below:

  ```
  # hiptest -I 0xmy_ifield 2097152 25
  ```

- To run the test when the system is attached to a switch, open up 2 UNIX shell windows. The windows can be on the same system or on different system running IRIS HIPPI. In one window, invoke *hiptest* as a receiver/destination (**-r** option). In the other window, invoke *hiptest* as a source (**-s** option). For the source, specify the destination's I-field. You must replace *dst_ifield* shown in the example with the value in the system's *hippi.imap* file that identifies the destination. When using a switch, it is recommended that the I-field have the camp-on bit set to one and the PS bits set to zero for source addressing.

  ```
  # hiptest -r
  # hiptest -s -I 0xdst_ifield
  ```

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

  ```
  # hiptest -I 0xmy_ifield_0 -D /dev/hippi0 &
  # hiptest -I 0xmy_ifield_1 -D /dev/hippi1 &
  # hiptest -I 0xmy_ifield_2 -D /dev/hippi2 &
  # hiptest -I 0xmy_ifield_3 -D /dev/hippi3 &
  ```

- To test four different IRIS HIPPI ports when they are attached to a switch, invoke four instances of the command in which 2 act as sources and 2 as receivers, as in the example below:

  ```
  # hiptest -s -I 0xdst_ifield_1 -D /dev/hippi0 &
  # hiptest -r -D /dev/hippi1 &
  # hiptest -s 0xdst_ifield_3 -D /dev/hippi2 &
  # hiptest -r -D /dev/hippi3 &
  ```

If the *hiptest* utility fails with an error message, locate the error message in the section “Alphabetical Error Message Listing” on page 99 and follow the instructions.

Verify a HIPPI Host-To-Host Interface

Once you have verified the loopback connection works properly, you can run a test between two hosts. You need to do run two copies of *hiptest* that have the same arguments except one has a **-r** and one has a **-s** command-line option.

On the destination host, enter:

```
hiptest -r
```

Then, on the source host, enter:

```
hiptest -I 0x01000009 -s 1024 10
```

for example, where `0x01000009` is the destination address that you are sending to. The source host should report that it has successfully sent 10 packets, and the destination host should report successfully receiving 10 packets.

Verify an IP-over-HIPPI Interface

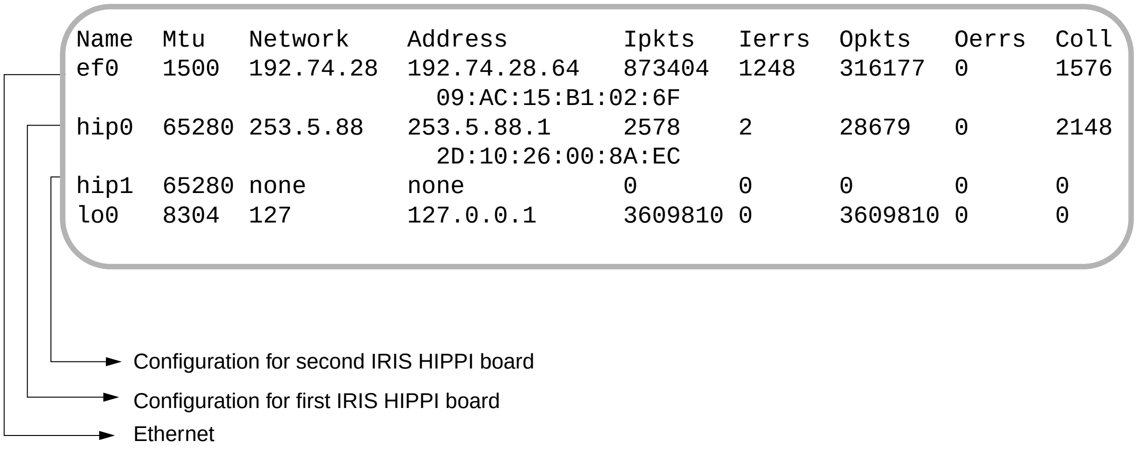
To verify that each IP-over-HIPPI network interface is functional, follow the instructions in this section. This test assumes that the HIPPI subsystem has passed the *hiptest* verification, as described under the heading “Verify the Board and Its HIPPI-FP Interface” in this section.

Note: Unlike many network software products, the IRIS HIPPI software does not loopback IP packets through the station’s local loopback interface (*lo0*). All IP-over-HIPPI packets are passed to the IRIS HIPPI hardware, regardless of destination address.

To accomplish this verification, use */usr/etc/ping -r* (lower case -r, not -R) to make this station communicate with another HIPPI IP station (or itself) over the HIPPI subsystem. If desired, you can use *hipcntl status* to monitor the packet and byte count of this test.

1. Obtain the IP network addresses for all the IRIS HIPPI boards on this system. This information can be displayed with the *netstat* command shown below, or with *hipmap -a*. The network address is listed in the column labelled `Network`, as illustrated in Figure 3-4.

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



```
Name  Mtu   Network    Address        Ipkts   Ierrs  Opkts   Oerrs  Coll
ef0   1500  192.74.28  192.74.28.64   873404  1248   316177  0      1576
                         09:AC:15:B1:02:6F
hip0  65280 253.5.88   253.5.88.1     2578    2      28679   0      2148
                         2D:10:26:00:8A:EC
hip1  65280 none       none           0       0      0       0      0
lo0   8304  127        127.0.0.1      3609810 0      3609810 0      0
```

Configuration for second IRIS HIPPI board

Configuration for first IRIS HIPPI board

Ethernet

Figure 3-4 The */usr/etc/netstat -ina* Display

2. Obtain the name (or IP address) of at least one station on each of these network addresses. Two methods for obtaining station names are described below.

   - For a system connected to a local area network that provides name lookup service (NIS), use the commands below to create a file of known hosts for each HIPPI network connection. Each file will contain the names and addresses of stations that share a particular network address:

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

     where *hip#_networkaddress* values are the addresses from the `Network` column of the *netstat* display (illustrated in Figure 3-4).

Example:

```
% ypcat hosts | grep 253.5.28 > hip0.s
```

- For a system that does not have access to NIS, use these commands to create a file of hosts that are known for each network connection. Each file will contain the locally-known names and addresses of stations that share a particular network address:

  ```
  % 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 located on the *hip0* network. For the variable *hip0_station*, you can use any of the names or IP addresses from the *hip0.s* file.

   ```
   % 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
   ```

   Note: If a loopback link is in place, use the system's own IP address for the *hip0_station* variable.

4. If *netstat* lists more than one IRIS HIPPI (*hip#*) network interface, 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.

   ```
   % 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
   ```

   Note: If a loopback link is in place on any of the ports, use the system's own IP address for the *hip#_station* variable.

5. If one *ping* on each network succeeds, you have completed the verification procedure. All the IRIS HIPPI 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 Board and Its HIPPI-FP Interface

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 configuration by performing the steps below.

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

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

2. Use */usr/etc/netstat -ina* to verify that the HIPPI network interfaces have been configured and enabled.

   Refer to the online *IRIX Admin: Networking* guide for information about configuring and troubleshooting IP.

3. If using static ARP, 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 names (or IP addresses). Verify that each entry is correct.
4. If using static ARP, verify that the */usr/etc/hippi.imap* file has correct entries for the local and remote I-fields.
5. If using HARP, use */usr/etc/hipmap* to verify the IP addresses and I-fields.
6. If using HIPPI source addressing, verify that each HIPPI cable is connected to the remote port that matches the HIPPI address (that is, the I-field).
7. If the system is connected to a switch, verify that the switch is operational.

If the *ping* verification tests succeed for one HIPPI network connection, but others fail, the IP stack is functioning, but one (or more) specific interfaces has a problem. To resolve the problem, follow the instructions below for each problematic network connection.

1. Make sure that you know which IRIS HIPPI board is associated with the HIPPI network interface (*hip*#) that you are troubleshooting.
2. Check that the HIPPI cables between the I/O panel and the other system (switch or endpoint) are tightly connected at both ends. Use the board's LEDs to verify that the physical link is functional.
3. If the system is connected to a switch, verify that the addresses and the physical ports on that switch are properly configured and operational.
4. If using static ARP, verify that the local */usr/etc/hippi.imap* file has an entry for the problematic local interface (that is, the IP address or name of the *hip*# interface) and for the remote hostname (or IP address) that failed.
5. Verify that the other endpoint (IP host) is operational.

   Or, as an alternate, select a different station on this LAN (network address). Try to *ping -r* that station using the numerical address (instead of the name). If the *ping* works, the network connection is functional. If the *ping* fails, proceed to the next step.
6. Verify that the network portion (leftmost digits) of the addresses you are attempting to *ping* match the local system's network address associated with the physical HIPPI connection you are troubleshooting. The local network address for each HIPPI network interface can be displayed by the */usr/etc/netstat -in* command.

Troubleshoot the HARP Server

Follow the procedure described in "Troubleshoot an IP Interface" on page 90, but follow steps on the HARP server as well as on those clients having trouble.

Note also that you should not be using the HARP server to perform infinite or very large transfers (connections lasting more than 5 minutes) because HARP information cannot be updated during a transfer.

Interpreting Status Information

The *hipcntl status* command displays a number of status and performance counts. Table 3-4 and Table 3-5 suggest how to interpret and use this information for troubleshooting different IRIS HIPPI hardware products. All of the counted items are initialized to zero upon reset of the board. Most of the counted items roll over to (start again at) zero upon reaching 2^{32} (that is, at 4,294,967,295); the exceptions are the bytecounts, which roll over at a little over 18 quintillion.

Table 3-4 hipcntl Command Status Information (Challenge and Onyx)

Item	Reasonable/Problematic Values
SRC connections:	Value should constantly increase, as long as local applications are sending data.
SRC packets:	Value should constantly increase, as long as local applications are sending data.
SRC rejects:	Count should be 0. The most probable cause for this event is incorrect configuration (for example, the I-field is incorrectly formatted or has an invalid setting, such as a W-bit set to indicate 64-bit-wide HIPPI).
SRC time outs:	Count should be 0. Each event indicates the remote system did not continue responding correctly after the initial connection was set up. This event can be caused by a timeout value that is too short.
SRC connects lost:	Count should be 0. Each event indicates a problem with a remote HIPPI system.
SRC parity errs:	Count should be 0. Each event indicates a problem with the local IRIS HIPPI hardware.
SRC seq errors (dm):	Count should be 0. Each event indicates a problem with the local hardware.
SRC seq errors (cd):	Count should be 0. Each event indicates a hardware problem with a remote HIPPI device.
SRC seq errors (cs):	Count should be 0. Each event indicates a problem with the local IRIS HIPPI hardware.
SRC dsic lost:	Count should be 0. Each event indicates a hardware or cabling problem located somewhere between the outbound port on the local panel plate connector and the inbound port on the adjacent HIPPI device (for example, the switch).

Table 3-4 (continued) hipcntl Command Status Information (Challenge and Onyx)

Item	Reasonable/Problematic Values
SRC number bytes sent:	Value should constantly increase, as long as local applications are sending data.
DST connections:	Value should constantly increase, as long as remote applications are sending data to this host.
DST packets:	Value should constantly increase, as long as remote applications are sending data to this host.
DST rcv on bad ulp:	Count should be 0. The most probable cause for this event is incorrect configuration (for example, the remote endpoint's HIPPI-FP layer is incorrectly configured or a local customer-developed application has not bound to its ULP-id correctly). Some programs can produce a few of these events when terminated unexpectedly (for example, doing a **Ctrl-C** to *hiptest*), because the receiver goes away before the transmitter.
DST hippi-le drop:	Count should be 0 or low. Usually caused by co-existence problems between IP and HIPPI-FP. For example, a local HIPPI-FP application may not be reading its input queue fast enough or it may be hogging the DMA engine with large-sized packets. Both conditions can result in the board's receive FIFO being blocked, so arriving IP packets are dropped. In rare instances, this condition can be caused by ULPs using very small-sized packets that results in more time spent on overhead processing than on data processing. In this case, increase the packet size (MTU) used by the sources.
DST llrc:	Count should be 0 or very low. The most probable cause is a high bit error rate along the physical link (for example, a dirty fiber end, a loose connector, a length of cable that is too long).
DST parity:	Count should be very low. The most probable cause for a high count is dirty fiber-optic ferrule tips, loose connectors, or too-long cabling.
DST null connections	May indicate a problem with a remote HIPPI system. Usually indicates that a remote application is opening and closing connections without sending data or has an intermittent hardware problem.
DST illegal burst:	Count should be 0. Each event indicates a hardware problem with a remote HIPPI device.
DST sequence err:	Count should be 0. Each event indicates a hardware problem with a remote HIPPI device.

Table 3-4 (continued) hipcntl Command Status Information (Challenge and Onyx)

Item	Reasonable/Problematic Values
DST sync err:	Count should be 0. Each event indicates that DST lost synchronization with its source.
DST sdic lost:	Count should be 0. Each event indicates a hardware or cabling problem located somewhere between the inbound port on the local panel plate connector and the outbound port on the adjacent HIPPI device (for example, the switch)
DST frame/state err:	Count should be 0. Each event indicates a hardware problem with a remote HIPPI device.
DST flag err:	Count should be 0 or very low. Each event indicates a hardware problem with a remote HIPPI device, except immediately following a local reset of the IRIS HIPPI hardware.
DST ready errors:	Count should be 0. Each event indicates a hardware problem with a remote HIPPI device.
DST bad packet starts:	Count should be 0. Each event indicates a hardware problem with a remote HIPPI device.
DST number bytes received	Value should constantly increase, as long as a remote system is sending data to this host.

Table 3-5 hipcntrl Command Status Information (Origin and Onyx2)

Item	Reasonable/Problematic Values
SRC connections:	Value should constantly increase, as long as local applications are sending data.
SRC packets:	Value should constantly increase, as long as local applications are sending data.
SRC rejects:	Count should be 0. The most probable cause for this event is incorrect configuration (for example, the I-field is incorrectly formatted or has an invalid setting, such as a W-bit set to indicate 64-bit-wide HIPPI).
SRC glink reset	Count should be 0. Each event indicates a problem with the local IRIS HIPPI hardware and/or cabling.

Table 3-5 (continued) hipcntrl Command Status Information (Origin and Onyx2)

Item	Reasonable/Problematic Values
SRC glink lost	Count should be 0. Each event indicates a problem with the local IRIS HIPPI hardware and/or cabling.
SRC time outs:	Count should be 0. Each event indicates the remote system did not continue responding correctly after the initial connection was set up. This event can be caused by a timeout value that is too short.
SRC connects lost:	Count should be 0. Each event indicates a problem with a remote HIPPI system.
SRC parity errs:	Count should be 0. Each event indicates a problem with the local IRIS HIPPI hardware.
SRC number bytes sent:	Value should constantly increase, as long as local applications are sending data.
DST connections:	Value should constantly increase, as long as remote applications are sending data to this host.
DST packets:	Value should constantly increase, as long as remote applications are sending data to this host.
DST rcv on bad ulp:	Count should be 0. The most probable cause for this event is incorrect configuration (for example, the remote endpoint's HIPPI-FP layer is incorrectly configured or a local customer-developed application has not bound to its ULP-id correctly). Some programs can produce a few of these events when terminated unexpectedly (for example, doing a **Ctrl-C** to *hiptest*), because the receiver goes away before the transmitter.
DST hippi-le drop:	Count should be 0 or low. Usually caused by co-existence problems between IP and HIPPI-FP. For example, a local HIPPI-FP application may not be reading its input queue fast enough or it may be hogging the DMA engine with large-sized packets. Both conditions can result in the board's receive FIFO being blocked, so arriving IP packets are dropped. In rare instances, this condition can be caused by ULPs using very small-sized packets that results in more time spent on overhead processing than on data processing. In this case, increase the packet size (MTU) used by the sources.
DST llrc:	Count should be 0 or very low. The most probable cause is a high bit error rate along the physical link (for example, a dirty fiber end, a loose connector, a length of cable that is too long).

Table 3-5 (continued) hipcntrl Command Status Information (Origin and Onyx2)

Item	Reasonable/Problematic Values
DST parity:	Count should be very low. The most probable cause for a high count is dirty fiber-optic ferrule tips, loose connectors, or too-long cabling.
DST frame/state err:	Count should be 0. Each event indicates a hardware problem with a remote HIPPI device.
DST flag err:	Count should be 0 or very low. Each event indicates a hardware problem with a remote HIPPI device, except immediately following a local reset of the IRIS HIPPI hardware.
DST illegal burst:	Count should be 0. Each event indicates a hardware problem with a remote HIPPI device.
DST link rdy lost in pkt:	Count should be 0. Each event indicates a problem with the local IRIS HIPPI hardware (for example, dirty fiber-optic end, loose connector, too-long or too-short cable, kinked/cracked cable) or possibly with a remote source. Use the various loopback tests to isolate the problem.
DST null connections	May indicate a problem with a remote HIPPI system. Usually indicates that a remote application is opening and closing connections without sending data or has an intermittent hardware problem.
DST ready errors:	Count should be 0. Each event indicates a hardware problem with a remote HIPPI device.
DST bad packet starts:	Count should be 0. Each event indicates a hardware problem with a remote HIPPI device.
DST number bytes received	Value should constantly increase, as long as a remote system is sending data to this host.

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.

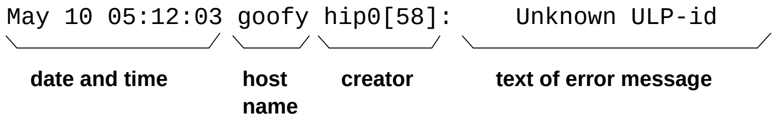
With each error message is a discussion of the problems the message may indicate. The list contains only messages that indicate an error or problem; it does not contain informational messages that occur during normal operation.

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 */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.



```
May 10 05:12:03 goofy hip0[58]:     Unknown ULP-id
```

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.

Common Implications of Error Reasons

When the following generic errors (errnos) are returned by IRIS HIPPI, the reason is usually due to the description below:

EBUSY The firmware on the board is not responding. It is probably hung; the message usually does not mean that the board is occupied with other work. This error can sometimes be remedied by doing a *hipcntl shutdown* followed by *hipcntl startup*.

EINVAL An invalid parameter or argument was given to the system call.

ENODEV The hardware is not operational or there is no device known by the supplied unit number. This error can sometimes be remedied by doing a *hipcntl shutdown* followed by *hipcntl startup*.

EPERM The process (or user) invoking the call did not have sufficient access rights. For example, the call may require superuser (root) privileges.

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.

```
FLAGS:   ACCEPTING DST.LNK_RDY DST.FSYNC DST.OH8SYNC DST.SIG_DET
```

When all of the flags are displayed (as shown above) by the *hipcntl status* command, the IRIS HIPPI-Serial port and its external cables are functional. If `ACCEPTING` is missing, use *hipcntl accept* to make the hardware start accepting connection requests. If `DST.SIG_DET` is present, but one of `DST.LNK_RDY`, `DST.FSYNC`, or `DST.OH8SYNC` is missing, the cable is probably dirty, not firmly connected, or damaged. For complete information about these flags, see the documentation for the panel plate LEDs.

`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 and "Common Implications of Error Reasons" on page 98.

```
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 and “Common Implications of Error Reasons” on page 98.

`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 and “Common Implications of Error Reasons” on page 98.

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

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 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 board is dysfunctional. Contact the Silicon Graphics Technical Assistance Center.

```
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 preceded 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. When the *reason* is `IO error`, this message probably means that an application has a file descriptor open for the device in question. Close all file descriptors for this device (for example, quit from Performance Co-Pilot). If all file descriptors are closed for this device, this message may indicate that the board is dysfunctional. Invoke *hipcntl shutdown* 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 and “Common Implications of Error Reasons” on page 98.

`hipmap: couldn’t bind socket:` *reason*

The utility was unable to bind to the raw socket. This indicates (1) a problem with the operating system (not with the IRIS HIPPI software or hardware) or (2) the IRIS HIPPI driver currently built into the operating system was configured to exclude support for the IP protocol stack. The *reason* is any of those described by the intro(2) man page and “Common Implications of Error Reasons” on page 98.

`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 and “Common Implications of Error Reasons” on page 98.

`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 and “Common Implications of Error Reasons” on page 98.

`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 and “Common Implications of Error Reasons” on page 98.

`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 and “Common Implications of Error Reasons” on page 98.

`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 and “Common Implications of Error Reasons” on page 98.

`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 and “Common Implications of Error Reasons” on page 98.

`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#) controlled 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: invalid maxsize` *user_input*

The value entered for packet size falls outside the minimum or maximum byte count supported.

`hiptest: invalid npkts` *user_input*

The value entered for number of packets falls outside the minimum or maximum supported.

`hiptest: read time-out`

The *hiptest* reading process failed to successfully read anything, so it timed out. There should be other error messages, in addition to this one, that explain the reason.

`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 and “Common Implications of Error Reasons” on page 98.

This indicates a problem with the software, too many applications trying to use the ULP-id, or a board that is shutdown. Perhaps the driver has not been built into the operating system or the IRIS HIPPI software has not been installed properly or the board has been shutdown with **`hipcntl shutdown`**. 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 and “Common Implications of Error Reasons” on page 98.

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*.

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#: 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): HIPPI DST errs: 0xerror_vector  error_text  error_text
```

When a *hiptest* **read()** call failed due to an EIO problem, *hiptest* retrieved the reasons for the failure from the board. The value of the returned 6-bit error vector is displayed in *error_vector* (in hexadecimal format). The reasons are provided in the *error_text* displays.

`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 byte counts 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, where 0 indicates the first packet. The *reason* is any of those described by the intro(2) man page and "Common Implications of Error Reasons" on page 98.

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

The *read()* call for the body of a packet failed, where a *packet#* of 0 indicates the first packet. The *reason* is any of those described by the intro(2) man page and "Common Implications of Error Reasons" on page 98.

`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 and "Common Implications of Error Reasons" on page 98.

`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 and "Common Implications of Error Reasons" on page 98.

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): HIPIOCW_ERR:` *error_num* *error_text*

When *hiptest*'s **write()** call failed, *hiptest* retrieved the reason for the failure from the board. The reason for the failure is provided in *error_num* (its ID) and *error_text*.

`hiptest(SRC): trouble doing HIPIOCW_ERR`

When *hiptest*'s **write()** call failed, *hiptest* tried to retrieve the reason for the failure from the board, but this attempt failed.

`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 and "Common Implications of Error Reasons" on page 98.

`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 for the timeout value are milliseconds entered in decimal format (for example, **`hipcntl stimeo 1000`** sets the timeout to 1000 milliseconds or 1 second).

Index

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 007-2229-006.

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