Rev 2.6

HIGH-PERFORMANCE PARALLEL INTERFACE - Memory Interface

(HIPPI-MI)

preliminary draft proposed
American National Standard
for Information Systems

March 13, 1992

Secretariat:

Computer & Business Equipment Manufacturers Association

ABSTRACT: The described Specification provides a Memory Interface protocol for the High-Performance Parallel Interface (HIPPI).

NOTE:

This is an internal working document of X3T9.3, a Task Group of Accredited Standards Committee X3. As such, this is not a completed standard. The contents are actively being modified by the X3T9.3 Task Group. This document is made available for review and comment only. For current information on the status of this document contact the individuals shown below:

POINTS OF CONTACT:

Don Tolmie (X3T9.3 Chairman and current Technical Editor)
Los Alamos National Laboratory
C-5, MS-B255
Los Alamos, NM 87545
(505) 667-5502
det@lanl.gov

Bob Beach (Original Technical Editor)
Ultra Network Technologies
101 Daggett Drive
San Jose, CA 95134
408-922-0100
bob@ultra.com

Foreword (This Foreword is not part of American National Standard X3.xxx-199x.)

This High-Performance Parallel Interface, Memory Interface (HIPPI-MI) standard defines a means to utilize the HIPPI as an interface into a memory subsystem. This specification has been designed around a minimum functionality, high performance model. It is intended to provide a basic set of primitives upon which more complex systems may be constructed. The HIPPI is designed for transmitting data at peak rates of 800 or 1600 Mbit/s between data processing equipment.

This standard was developed by Task Group X3T9.3 of Accredited Standards Committee X3 during 1989, 1990, and 1991. The standards approval process started in 1991.

Requests for interpretation, suggestions for improvement or addenda, or defect reports are welcome. They should be sent to the X3 Secretariat, Computer and Business Equipment Manufacturers Association, 311 First Street, Suite 500, Washington, DC 20001-2178.

This standard was processed and approved for submittal to ANSI by Accredited Standards Committee on Information Processing Systems, X3. Committee approval of the standard does not imply that all committee members voted for approval. At the time it approved this standard, the X3 Committee had the following members:

Richard Gibson, Chair
Donald C. Loughry, Vice-Chair
Joanne M. Flanagan, Secretary

Organization Represented	*Name of Representative*
3M Company	Paul D. Jahnke
AICCP	Kenneth Zemrowski
	Eugene M. Dwyer (Alt)
Allen-Bradley Company	Ronald H. Reimer
	Joe Lenner (Alt)
American Library Association	Paul E. Peters
American Nuclear Society	Geraldine C. Main
	Sally Hartzell (Alt)
AMP Incorporated	Edward R. Kelly
	Edward Mikoski (Alt)
Apple Computer, Inc.	Karen Higginbottom
AT&T Corporation	Thomas F. Frost
	Paul D. Bartoli (Alt)
Boeing Co.	Catherine Howells
	Andrea Vanosdoll (Alt)
Bull HN Information Systems.	David M. Taylor
Compaq Computer Corporation	James L. Barnes
	Keith Lucke (Alt)
DECUS	Stephen C. Jackson
	Dr. Joseph King (Alt)
Digital Equipment Corporation	Gary S. Robinson
	Del Shoemaker (Alt)
Eastman Kodak Co.	James D. Converse
	Michael Nier (Alt)
Electronic Data Systems Corporation	Jerrold S. Foley
	Charles M. Durrett (Alt)
General Services Administration	Douglas K. Arai
	Larry L. Jackson (Alt)
GUIDE International	Frank Kirshenbaum
	Harold Kuneke (Alt)
Hewlett-Packard Company	Donald C. Loughry
Hitachi America Ltd.	Kei Yamashita
	John Neumann (Alt)
Hughes Aircraft Co.	Harold L. Zebrack
IBM Corporation	Robert H. Follett
	Mary Anne Gray (Alt)
Lawrence Berkeley Laboratory	David F. Stevens
	Robert L. Fink (Alt)
National Communications System	Dennis Bodson
	George W. White (Alt)
National Institute of Standards & Technology	Robert E. Rountree
	Michael D. Hogan (Alt)
NCR Corporation	Thomas W. Kern
	A. R. Daniels (Alt)
Omnicom Inc	Harold C. Folts
	Kathleen Dally (Alt)

Organization Represented	*Name of Representative*
Open Software Foundation	Fritz Schulz
Recognition Technology Users Association	Herbert F. Schantz
	G. Edwin Hale (Alt)
SHARE, Inc.	Thomas B. Steel Jr.
	Gary Ainsworth (Alt)
Sony Corp.	Michael Deese
Storage Technology Corp	Joseph S. Zajaczkowski
	Samual D. Cheatham (Alt)
Sun Microsystems Inc.	Scott K. Jameson
Unisys Corporation	Stephen P. Oksala
	John L. Hill (Alt)
U.S. Department of Defense	William C. Rinehuls
	Thomas E. Bozek (Alt)
U.S. Department of Energy	Alton Cox
	John Ruatto (Alt)
US West Corp	Gary Dempsey
	Susan Capraro (Alt)
USE Inc	Pete Epstein
Wang Laboratories, Inc	Steve Brody
	Sarah Wagner (Alt)
Wintergreen Information Services	John L. Wheeler
Xerox Corporation	Roy Pierce

Subcommittee X3T9 on Computer Input/Output Interfaces, which reviewed this standard, had the following members:

Delbert L. Shoemaker, Chair
Robert L. Fink, Vice-Chair

Charles Brill
William Burr
Stephen Cooper
Roger Cummings
James Hamstra
Reinhard Knerr
Lawrence Lamers
John Lohmeyer
Bruce McClure
George Michael
Arthur Miller
Gene Milligan
Steven Myers
Roger Pandanda
Everett Rigsbee
Floyd Ross
Amit Shah
Geoffrey Stone
Mary Jane Strohl
Don Tolmie
Schelto van Doorn
Jim Vogt
Carl Zeitler, Jr.

Task Group X3T9.3 on Device Level Interfaces, which was responsible for the development of this standard, had the following principal and alternate members:

Don Tolmie, Chair
Roger Cummings, Vice-Chair

D. Allan
R. Anderson
T. Anderson
K. Annamalai
M. Baird
C. Barron
B. Beach
A. Bedi
B. Bellino
H. Brandt
C. Brill
M. Carboni
B. Carlson
J. Connell
B. Cornelius
R. Doolittle
P. Dougherty
S. Erler
G. Frederickson
M. Friedmann
S. Gillett
J. Goell
E. Grivna
J. Guedj
D. Gustafson
D. Guss
M. Halvorson
K. Hardwick
M. Hartmann
M. Henske
D. Hepner
J. Huber
J. Hughes
R. Johnson
T. Kaminski
J. Kerr
R. Knerr
J. Kubinec
P. Lambert
P. Lichtenberger
R. Liu
J. Luttrull
K. Malavalli
T. McClendon
G. Michael
G. Milligan
B. Morris
J. Morris
D. Morrissey
R. Niemi
M. O'Donnell
T. Palkert
V. Patel
N. Perdue
G. Phillips
R. Phillips
P. Ramsey
E. Reid
J. Renwick
W. Roiger
P. Rupert
T. Russell
T. Salthouse
W. Sanderson
P. Savage
P. Schreck
J. Schuessler
P. Scott
C. Seeley
K. Sivakumar
J. Smith
D. Somes
J. Soriano
R. Southard
G. Stager
A. Stone
T. Szostak
R. Taborek
R. Thomsen
H. Todd
C. Towndrow
H. Truestedt
S. van Doorn
L. Veil
M. Vogel
Y. Wang
L. Whitfield
S. Zanowick

Contents

Page

Foreword iii

Introduction vi

1 Scope 1

2 Normative references 1

3 Definitions and conventions 1
- **3.1** Definitions 1
- **3.2** Editorial conventions 2

4 Elements of HIPPI-MI 2
- **4.1** Conceptual overview 2
- **4.2** Access control 2
- **4.3** Source system identifiers 2
- **4.4** Target system identifiers 3
- **4.5** Initialization sequence 3
- **4.6** Read sequence 4
- **4.7** Write sequence 4
- **4.8** Lock sequence 5
- **4.9** Message sequence 5
- **4.10** Cancel sequence 6
- **4.11** No_Operation sequence 6
- **4.12** Use of other HIPPI-FP features 6

5 Service interface primitives 7
- **5.1** Overview of service primitives 7
- **5.2** Primitives and packet relationship 8
- **5.3** HIPPI-MI service primitives summary 8
- **5.4** Initialize primitives 8
- **5.5** Read data primitives 9
- **5.6** Write data primitives 10
- **5.7** Lock primitives 11
- **5.8** Message primitives 13
- **5.9** Handshake primitives 14

6 HIPPI-FP interface to HIPPI-MI 15

7 HIPPI-MI packet formats 16
- **7.1** Common fields 16
- **7.2** Initialization command information fields 17
- **7.3** Initialization reply information fields 17
- **7.4** Read, write, and lock parameter fields 18
- **7.5** Lock information fields 18
- **7.6** Message information fields 18

Figures

Figure 1- Control hierarchy vi
Figure 2 - Memory interface model 2
Figure 3 - Service primitive components 7
Figure 4 - Read, write, lock and initialize service primitive sequences 7
Figure 5 - Message service primitive sequence 13
Figure 6 - HIPPI-MI initialization command packet format 19
Figure 7 - HIPPI-MI initialization reply packet format 19
Figure 8 - HIPPI-MI read and cancel command packet formats 20
Figure 9 - HIPPI-MI read and cancel reply packet formats 20
Figure 10 - HIPPI-MI write and No_Operation command packet formats 21
Figure 11 - HIPPI-MI write and No_Operation reply packet formats 21
Figure 12 - HIPPI-MI lock command packet format 22
Figure 13 - HIPPI-MI lock reply packet format 22
Figure 14 - HIPPI-MI message command packet format 23
Figure 15 - HIPPI-MI message reply packet format 23

Introduction

This High-Performance Parallel Interface, Memory Interface (HIPPI-MI) standard defines a protocol and formats by which an external device may gain access to a memory subsystem of another device. This external device passes commands and data across the HIPPI channel to the memory subsystem, allowing the external device read/write access to the contents of the memory.

Characteristics of this HIPPI memory interface include:

- Minimum functionality with simple read, write, lock, and message commands supported.
- Very simple sequences and formats that may be implemented in hardware.
- Defined as a HIPPI-FP Upper Layer protocol.
- Operates in either a point-to-point or switched configuration.

The information in the document is organized as follows:

- Clause 1 provides the introductory material.
- Clause 2 lists the publications referenced in this standard.
- Clause 3 provides a glossary and editorial conventions.
- Clause 4 specifies the basic elements of the HIPPI-MI.
- Clause 5 specifies the service interface between the HIPPI-MI and the user, and the primitive operations.
- Clause 6 lists the HIPPI-FP primitives used to interface the HIPPI-MI to the HIPPI Framing Protocol. (HIPPI-FP)
- Clause 7 specifies the parameter and packet formats.

Figure 1 shows the relationship of this standard (in the solid rectangle) to the other entities shown.

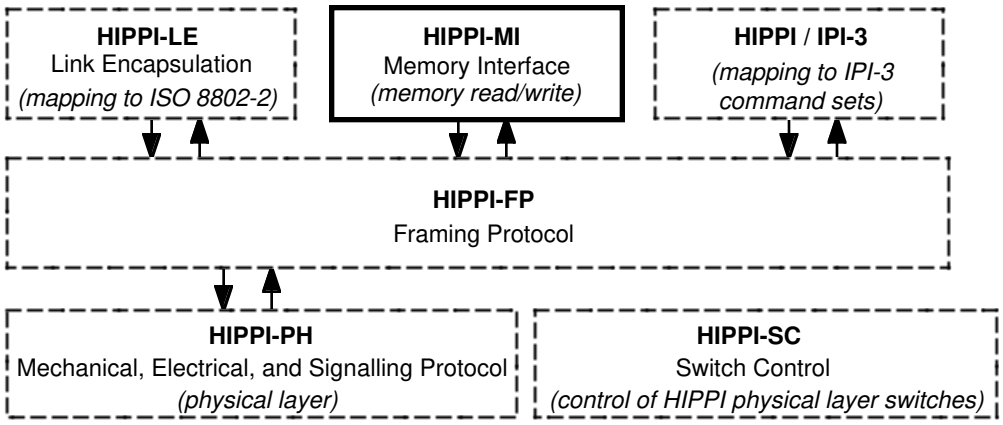

Figure 1 - Control hierarchy

Information processing - High-Performance Parallel Interface -

Memory Interface (HIPPI-MI)

1 Scope

This American National Standard defines a protocol and formats by which an external device may gain access to a memory subsystem of another device via the X3.210 HIPPI Framing Protocol (HIPPI-FP).

The purpose of this standard is to facilitate the development and use of the HIPPI in computer systems by providing a means for closely coupled computer systems to gain access to each other's memory subsystems. The facility will be used as a basis for a number of system configurations including shared memory devices, attached coprocessors, multi-computer complexes, and so forth.

2 Normative references

The following standards contain provisions which, through reference in this text, constitute provisions of this standard. At the time of publication, the editions indicated were valid. All standards are subject to revision, and parties to agreements based on this standard are encouraged to investigate the possibility of applying the most recent editions of the standards listed below. Members of IEC and ISO maintain registers of currently valid International Standards.

ANSI X3.210-199x, *High-Performance Parallel Interface, Framing Protocol Specification (HIPPI-FP)*.

ANSI X3.222-199x, *High-Performance Parallel Interface - Switch Control (HIPPI-SC)*.

3 Definitions and conventions

3.1 Definitions

For the purposes of this standard, the following definitions apply.

3.1.1 Access_Key: An eight-byte field that enables a user to execute a HIPPI-MI command.

3.1.2 burst: A group of words sent by the source to the destination. Bursts contain 1 to 256 words. Bursts that contain less than 256 words are called short bursts. A packet contains no more than one short burst. A short burst may be either the first or last burst of a multiburst packet.

3.1.3 byte: A group of eight bits.

3.1.4 connection: Condition of the HIPPI-FP when data transfers from source to destination are possible.

3.1.5 connection control information (CCI): A parameter sent as part of the sequence of operations establishing a connection from a source to a destination.

3.1.6 destination: The equipment that receives the data.

3.1.7 lock: An indivisible read, inspect, and write sequence to a memory location in the target system.

3.1.8 Lockword: A four-byte, or eight-byte, unit of information in the target system's memory which is immediately affected by a lock operation. The state of a Lockword may control access to a much larger memory entity such as a data buffer.

3.1.9 message: A command directed to an object, or a response from an object.

3.1.10 memory subsystem: An entity that contains addressable computer memory.

3.1.11 object: An intelligent entity in the target system to which messages are passed.

3.1.12 optional: Features that are not required by the standard. However, if any optional feature defined by the standard is implemented, it shall be implemented according to the standard.

3.1.13 packet: A data set sent from a HIPPI source to a HIPPI destination. A packet is composed of one or more bursts.

3.1.14 service interface: Connection points to the HIPPI-MI service user.

3.1.15 source system: The equipment that generates HIPPI-MI commands.

3.1.16 target system: The equipment that replies to HIPPI-MI commands.

3.1.17 User_Key: A string of up to 64 bytes that identifies the user during an initialization sequence.

3.2 Editorial conventions

In this standard, certain terms that are proper names of primitives, or similar terms are printed in uppercase to avoid possible confusion with other uses of the same words (e.g., MI_READ_CMD). Any lowercase uses of these words have the normal technical English meaning.

A number of conditions, sequences parameters, events, English text, or similar terms are printed with the first letter of each word in uppercase and the rest lowercase (e.g., Access_Key). Any lowercase uses of these words have the normal technical English meaning.

4 Elements of HIPPI-MI

4.1 Conceptual overview

This specification defines a method by which an external device gains access, via HIPPI-FP operations, to a memory subsystem in another machine. The physical level interconnection may be either HIPPI-PH, as specified in X3.183, or equivalent. Figure 2 shows the general model consisting of:

- a source system with a bidirectional pair of HIPPI-PH interfaces,
- a target system with some type of addressable memory and a bidirectional pair of HIPPI-PH interfaces, and
- two interconnecting HIPPI-PH channels.

The source system generates initialize, read, write, lock, and message commands, packages them in command packets, and passes them as HIPPI packets over the HIPPI channel to the target system. The target system performs the requested operation upon the indicated memory. For read operations, the requested data is sent back in a reply packet to the source system over the HIPPI channel.

In figure 2 there are several memory buffers present in the target system. The source system is shown writing "n" bytes to a memory area (Memory Buffer #1) in the target system starting at address "m" .

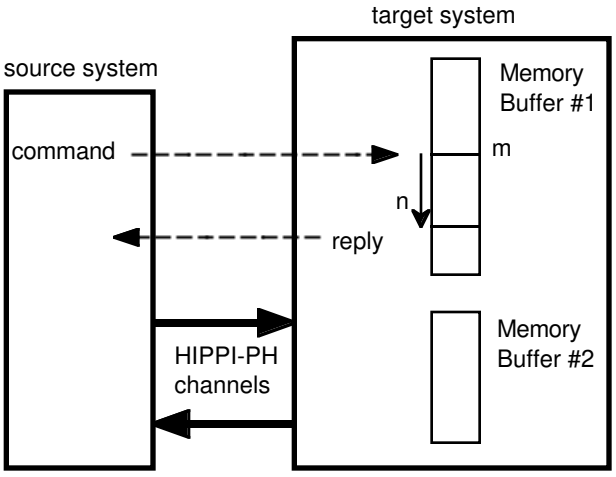

Figure 2 - Memory interface model

The MI sequences for reading from, writing to, and executing lock operations on, a target system memory are structured to be simple so that they may be processed quickly by either hardware or software.

This standard addresses only basic read, write, lock, and message operations, and an initialization sequence, between a source system and target system pair. It does not address higher level issues such as consistency among multiple readers/writers, protection, or scatter/gather. These are important issues that may need to be addressed in various applications; however, they are considered to be beyond the scope of this standard.

Not all operations may be supported on all systems. The initialize sequence may be used to determine the capabilities of a target system.

4.2 Access control

HIPPI-MI provides a means for insuring only authorized access to the memory of the target system. The means to provide this access control is an 8-byte Access_Key that serves as a "key" to the target system. Systems attempting an MI read, write, lock, or message operation shall provide an Access_Key in the command packet. The value for the Access_Key field is provided during the initialization sequence. A command with an improper Access_Key shall be rejected with an access denied error indication.

To obtain an Access_Key the source system shall provide a User_Key of up to 64 bytes to the target system. The User_Key is conceptually a "password" to the target system. If the request is acceptable to the target system, the target system shall provide an Access_Key to the source system.

The User_Key is a long string that may take considerable time to verify and hence is only submitted during the initialization sequence. The Access_Key is much shorter and is required with every command. The User_Key may also contain information on service type requested, number and size of buffers requested, etc. The formats of the Access_Key and User_Key, and the mapping between them, is implementation specific and is not specified in this standard. It is suggested, but not required, that the Access_Key be randomly generated.

4.3 Source system identifiers

The source system may use the source identifier (16-bit S_id_1 and 32-bit S_id_2 and S_id_3) parameters to associate a command, and its reply, with a process, processor, sequence number, upper-layer protocol, or other entity in the source system. Each command issued by the source system shall contain the S_id_1, S_id_2 and S_id_3 (also written as S_id_1,2,3) parameters. These parameters shall be returned, unmodified, when the target system issues a reply to the command. The format and partitioning of the S_id_1,2,3 parameters is implementation specific and is not specified in this standard.

The source system may request, via the Interrupt parameter, that the source system be signaled, e.g., interrupted, when the reply to a command is received. When the target system sends a reply packet, it shall set the value of the Signal parameter in the reply packet equal to the value of the Interrupt parameter in the command packet. The Interrupt parameter in the reply packet shall be the same as in the command packet.

4.4 Target system identifiers

The memory subsystem of the target system is defined as consisting of one or more buffers, each containing up to 2^{32} bytes. Each buffer shall be identified by a 32-bit Target Buffer Identifier (T_Buffer_id). A particular memory location shall be selected by the T_Buffer_id and a 32-bit byte offset (T_First_Byte_Offset). It is not required that there be only one T_Buffer_id/T_First_Byte_Offset mapping for a particular location of actual memory. It may be that any particular location can be selected by a large number of T_Buffer_id/T_First_Byte_Offset combinations.

The format of T_Buffer_id, and its mapping to memory addresses, is implementation specific and is not specified in this standard. Depending on the implementation a T_Buffer_id may be an address, an index into a table, a "pointer", etc. In general, systems issuing HIPPI-MI commands should treat the T_Buffer_id as a simple token and not presume any additional semantics.

T_First_Byte_Offset is defined to be a byte offset relative to the start of the buffer in the target system. T_First_Byte_Offset = 0 points to the first byte of the buffer. Systems have the option of restricting the granularity to be a multiple of a specified number of bytes. Both sides of a HIPPI-MI connection are informed of this restriction during the initialization sequence. If a target system receives a command with a T_First_Byte_Offset or Transfer_Length value that does not conform to its access granularity, it shall reject such commands with an invalid request error indication. Regardless of the T_First_Byte_Offset and Transfer_Length parameters supplied by the source system, the target system shall not transfer information to or from areas beyond the extent of the buffer identified by the T_Buffer_id.

HIPPI-MI allows non-zero D2_Offset values (see HIPPI-FP document). If a target system cannot support non-zero D2_Offset values, it shall reject such commands with an invalid request error indication. If the D2_Offset field is non-zero, then the target system receiving such a packet shall discard the offset bytes in the D2_Area before beginning the transfer of valid bytes into its memory subsystem. The ability to use a non-zero D2_Offset is indicated in the initialization sequence.

HIPPI-FP supports starting the D2_Data_Set (see HIPPI-FP document) on a burst boundary or within the first burst. The target system's preference is indicated in the initialization sequence.

The Signal parameter is used to notify, e.g., interrupt, the destination when the destination packet recipient completes the command.

4.5 Initialization sequence

The initialization sequence is optional, implicit initialization is permitted. The initialization sequence consists of a simple handshake in which a source system notifies the target system of its desire to access the target system's memory subsystem. The goals of the initialization sequence are:

- For the source system to obtain an Access_Key enabling read, write, lock, and message commands. To do this, the source system provides a User_Key and the target system returns the Access_Key.

- For the source system and target system to agree upon the features of the MI protocol that shall be used. For example, the use of the D2_Offset, starting the D2_Data_Set in the first or second burst, restrictions on access granularity, etc.

In order not to complicate the implementation of the basic read, write, and lock operations, the initialization and message packets use a different HIPPI-FP ULP-id than is used with the other packets.

An initialize command packet shall contain the following information:

- Operation = Initialize command;
- S_id_1,2,3: source system identifiers;
- Signal: if the target system shall be notified when the transfer is complete;
- Interrupt: if the source system shall be notified when the reply is received;
- Transfer_Length: the number of bytes plus four (for the S_Parameters and T_Parameters) in the User_Key;
- Access_Key = zeros: ignored;
- T_Buffer_id = zeros; ignored;
- T_First_Byte_Offset = zeros: must be zero;
- T_Parameters = zeros: space for target to fill in values;
- S_Granularity: the minimum granularity of the T_First_Byte_Offset and Transfer_Length that could be requested by the source system;
- Zero: source system requires zero D2_Offsets in reply packets;
- Start_D2_on_Burst: source system requires the read data in read reply packets from the target system to start in the second burst; and
- User_Key: information identifying the source system and user.

An initialize reply packet shall be issued by the target system in reply to an initialize command. An initialize reply packet shall contain the following information:

- Operation = Initialize reply;
- S_id_1,2,3: source system identifiers;
- Signal: if the source system shall be notified when the reply is received;
- Interrupt: if the source system shall be notified when the reply is received;
- Transfer_Length = 4: the length of the S_Parameters and T_Parameters
- Access_Key: Access_Key value to be used by source system user for subsequent operations (see 4.2);
- T_Buffer_id = zeros; ignored;
- T_First_Byte_Offset = zeros: must be zero;
- S_Parameters: the same as in the Initialization Command packet;
- Eight: target system supports 8-byte Lockwords;

- Four: target system supports 4-byte Lockwords;
- Message: target system supports message commands;
- Write: target system supports write commands;
- Read: target system supports read commands;
- Start_D2_on_Burst: target system requires the write data in write command packets to start in the second burst;
- Zero: target system requires zero D2_Offsets in reply packets;
- T_Granularity: the minimum granularity of the T_First_Byte_Offset and Transfer_Length supported by the target system; and
- MI_Status: indication of success, or nature of failure.

The initialize reply packet the S_id_1,2,3, and Interrupt parameters shall contain the same values as supplied by the source system in the initialize command packet. The value of the Signal parameter in the initialize reply packet shall be the same as the value of the Interrupt parameter in the initialize command packet.

4.6 Read sequence

A read command packet shall contain the following information:

- Operation = Read command;
- Access_Key: see 4.2;
- S_id_1,2,3: source system identifiers;
- T_Buffer_id: target system buffer identifier;
- T_First_Byte_Offset: relative location of first byte to be read;
- Transfer_Length: requested number of bytes to be transferred;
- Signal: if the target system shall be notified when the transfer is complete; and
- Interrupt: if the source system shall be notified when the reply is received.

The T_First_Byte_Offset and Transfer_Length shall conform to the granularity parameters obtained during an initialization. No data is passed with the read command.

A read reply packet shall be issued by the target system in reply to the read command. A read reply packet shall contain the following information:

- Operation = Read reply;
- Access_Key: see 4.2;
- S_id_1,2,3: source system identifiers;
- T_Buffer_id: target system buffer identifier;
- T_First_Byte_Offset: relative location of first byte to be read;
- Transfer_Length: number of bytes actually transferred;
- Signal: if the source system shall be notified when the transfer is complete;
- Interrupt: if the source system shall be notified when the transfer is complete;
- Data: the data read from the target system;
- MI_Status: indication of success, or nature of failure.

The read reply packet Access_Key, S_id_1,2,3, T_Buffer_id, T_First_Byte_Offset, Transfer_Length, and Interrupt parameters shall contain the same values as supplied by the source system in the read command packet. The value of the Signal parameter in the read reply packet shall be the same as the value of the Interrupt parameter in the read command packet. Note that the Transfer_Length specifies the number of bytes actually transferred, and may be less than the number of bytes requested.

4.7 Write sequence

A write command packet shall contain the following information:

- Operation = Write command;
- Access_Key: see 4.2;
- S_id_1,2,3: source system identifiers;
- T_Buffer_id: target system buffer identifier;
- T_First_Byte_Offset: relative location of first byte to be written;
- Transfer_Length: maximum number of bytes to be transferred;
- Signal: if the target system shall be notified when the transfer is complete;
- Interrupt: if the source system shall be notified when the reply is received;
- Answer: the target system should issue a reply; and
- Data: the data to be written in the target system buffer.

Note that Transfer_Length in the write command packet specifies the maximum number of bytes to be transferred, the actual number of bytes transferred may be less than this value. The Transfer_Length must also conform to the granularity parameters obtained during an initialization.

If there are no errors, and if a write reply was requested (with the Answer parameter = 1) in the write command, then the target system shall generate a write reply packet and send it to the source system. If there are errors a write reply shall be generated, regardless of whether it was requested or not, and shall indicate the nature of the error. A write reply packet shall contain the following information:

- Operation = Write reply;
- Access_Key: see 4.2;
- S_id_1,2,3: source system identifiers;
- T_Buffer_id: target system buffer identifier;
- T_First_Byte_Offset: relative location of first byte to be written;
- Transfer_Length: number of bytes actually written to the target system;
- Signal: if the target system shall be notified when the transfer is complete;
- Interrupt: if the source system shall be notified when the reply is received;
- Answer: the target system was requested to issue a reply; and
- MI_Status: indication of success, or nature of failure.

The write reply packet Access_Key, S_id_1,2,3, T_Buffer_id, T_First_Byte_Offset, Interrupt, and Answer parameters shall contain the same values as supplied by the source system in the write command packet. The value of the Signal parameter in the write reply packet shall be the same as the value of the Interrupt parameter in the write command packet. Note that the Transfer_Length specifies the number of bytes actually written to the target system memory subsystem, and may not be the same as the number of bytes requested in the write command packet.

4.8 Lock sequence

Applications exchanging information between source and target systems often encounter a difficulty seen in multi-processor systems, i.e., they must check for the availability of a resource in memory and reserve it in a single memory operation, preventing confusion if another processor is performing the same test at the same time. Lock operations provide a means for the source and target systems to pass ownership of buffers and control areas back and forth by providing indivisible read, inspect, and write operations.

Lock commands and replies strongly resemble read and write operations, and provide many of the same services. Lock commands operate on a small area of storage known as "Lockwords". HIPPI-MI supports Lockwords which are either four or eight bytes in length. A Lockword is usually the key to a much larger memory resource such as a buffer. The state of the Lockword may indicate if the buffer is generally available, or is reserved for the exclusive use of the source or target system. The format of the Lockwords is implementation specific and is not specified in this standard.

Four data elements are associated with lock operations:

- Old_Data: the value in the target system's Lockword before the lock operation begins. All lock operations shall return the Old_Data to the source system in a lock reply packet.

- Check_Value: a value provided by the source system to compare against the Old_Data.

- New_Data: the value provided by the source system to replace or modify Old_Data.

- Test: a binary value returned by the target system which shall indicate if the Check_Value equals the Old_Data.

The HIPPI-MI service defines three types of lock operations. Support of lock operations is optional on a target system, but a system which supports locks shall support all three operations. The lock operations are:

- Swap: The target system shall unconditionally store the New_Data value in the Lockword and shall return the previous value of the Lockword as Old_Data in the reply.

- Compare and Swap: The target system shall compare the Check_Value with the Old_Data. If equal, the target system shall replace the Old_Data with the New_Data and shall return the Old_Data in the reply. If unequal, the target system shall leave the target system memory contents unmodified and shall return the Old_Data in the reply.

- Fetch and Add: The target system shall unconditionally add the contents of the New_Data to the Old_Data, storing the sum in the Lockword. The target system shall return the previous value of Old_Data in the reply.

A lock command packet shall contain the following information:

- Operation = Lock ... command;
- Access_Key: see 4.2;
- S_id_1,2,3: source system identifiers;
- T_Buffer_id: target system buffer identifier;
- T_First_Byte_Offset: relative location of the first byte of the Lockword;
- Transfer_Length: twice the number of bytes in the Lockword (8 or 16);
- Signal: if the target system shall be notified when the transfer is complete;
- Interrupt: if the source system shall be notified when the reply is received;
- Operation: lock operation to be performed;
- Check_Value: source system value to be compared; and
- New_Data: modify the target system Lockword with this value if the operation is successful.

A lock reply packet shall be issued by the target system in reply to the lock command. A lock reply packet shall contain the following information:

- Operation = Lock ... reply;
- Access_Key: see 4.2;
- S_id_1,2,3: source system identifiers;
- T_Buffer_id: target system buffer identifier;
- T_First_Byte_Offset: relative location of the first byte of the Lockword;
- Transfer_Length: twice the number of bytes in the Lockword (8 or 16);
- Signal: if the source system shall be notified when the reply is received;
- Interrupt: if the source system shall be notified when the reply is received;
- Test: the result of the lock operation;
- Operation: lock operation that was performed;
- Check_Value: source system value to be compared;
- Old_Data: original contents of Lockword before operation was executed; and
- MI_Status: indication of success, or nature of failure.

The lock reply packet Access_Key, S_id_1,2,3, T_Buffer_id, T_First_Byte_Offset, Transfer_Length, Interrupt, and Check_Value parameters shall contain the same values as supplied by the source system in the lock command packet. The value of the Signal parameter in the lock reply packet shall be the same as the value of the Interrupt parameter in the lock command packet.

4.9 Message sequence

Message passing is provided to pass commands to an object in the target system, e.g., commands to a co-processor. Message sequences are composed of message command and reply packets. These operations use the same ULP-id as the initialization and cancel commands.

A message command packet shall contain the following information:

- Operation = Message command;
- Access_Key: see 4.2;
- S_id_1,2,3: source system identifiers;
- Object_id: identifies the specific object in the target system, e.g., a specific processor of a multi-processor system, or a specific process, etc.;
- Transfer_Length: number of bytes of Message_Data;
- Signal: if the target system shall be notified when the transfer is complete;
- Interrupt: if the source system shall be notified when the reply is received;
- Message_Operation; operation to be performed; and
- Message_Data: parameters for the operation (if needed for this operation).

The Transfer_Length shall conform to the granularity parameters obtained during an initialization.

A message reply packet shall be issued by the target system in reply to the message command. A message reply packet shall contain the following information:

- Operation = Message reply;
- Access_Key: see 4.2;
- S_id_1,2,3: source system identifiers;
- Object_id: identifies the specific object in the target system;
- Transfer_Length: number of bytes of Message_Reply_Data;
- Signal: if the source system shall be notified when the transfer is complete;
- Interrupt: if the source system shall be notified when the transfer is complete;
- Message_Operation; operation performed;
- Message_Reply_Data: response parameters (if needed for this operation); and
- MI_Status: indication of success, or nature of failure.

The message reply packet Access_Key, S_id_1,2,3, Object_id, and Interrupt parameters shall contain the same values as supplied by the source system in the message command packet. The value of the Signal parameter in the message reply packet shall be the same as the value of the Interrupt parameter in the message command packet.

4.10 Cancel sequence

A cancel sequence may be used to retire an Access_Key and buffers associated with the Access_Key. A cancel sequence is equivalent to an un-initialize or close operation.

The cancel sequence uses the MI_READ_xxx primitives of 5.5 and read packet format shown in figures 8 and 9. A cancel command packet shall contain the following information. Differences between a read command packet and a cancel command packet are marked with a *.

- Operation = Cancel command (*);
- Access_Key: identifies the Access_Key to be canceled (*);
- S_id_1,2,3: source system identifiers;
- T_Buffer_id: ignored (*);
- T_First_Byte_Offset: ignored (*);
- Transfer_Length: ignored (*);
- Signal: if the target system shall be notified when the transfer is complete; and
- Interrupt: if the source system shall be notified when the reply is received.

A cancel reply packet shall be issued by the target system in reply to the cancel command. A cancel reply packet shall contain the following information:

- Operation = Cancel reply (*);
- Access_Key: identity of the Access_Key that was canceled (*);
- S_id_1,2,3: source system identifiers;
- T_Buffer_id: ignored (*);
- T_First_Byte_Offset: ignored (*);
- Transfer_Length: ignored (*);
- Signal: if the source system shall be notified when the transfer is complete;
- Interrupt: if the source system shall be notified when the transfer is complete;
- Data: no data is transferred from the target system (*); and
- MI_Status: indication of success, or nature of failure.

4.11 No_Operation sequence

The No_Operation sequence uses the same packet formats and primitives as the write sequence (see 5.6), except with a No_Operation operation code (see 7.1) and no data field. The values for all of the parameters in the No_Operation command packet, except for the Operation and Answer parameters, shall be ignored by the target system.

A No_Operation reply packet shall be generated by the target system if so requested with Answer = 1 in the No_Operation command packet. All of the parameters in the No_Operation reply packet, with the exception of the Signal and Operation parameters, shall be an echo of those in the No_Operation command packet.

4.12 Use of HIPPI-FP features

HIPPI-MI uses both the D1_Area and D2_Area of the HIPPI-FP. The data, if present, shall be contained in the D2_Data_Set in the D2_Area. See figures 6 - 15 for packet layouts. The S_id_3 field shall be the last information in the HIPPI-FP D1_Data_Set; any following information, e.g., data, Check_Value, Message_Data, or S_Parameters, shall be contained in the D2_Data_Set within the D2_Area.

The return route from the target system to the source system may be determined from the CCI received from the HIPPI-FP in the target system. See the X3.222 HIPPI-SC document for examples of possible CCI translations.

5 Service interface primitives

5.1 Overview of service primitives

The services described by the following set of named primitives are those required for the correct operation of the HIPPI-MI. How many of the services described herein are chosen for a given implementation is up to the implementer; however, a set of HIPPI-MI services must be supplied sufficient to satisfy the user. All of the listed primitives and parameters are considered as required except where explicitly stated otherwise. The services herein do not imply any particular implementation, or any interface.

The service interface is the collection of primitive operations by which the HIPPI-MI provides services to the user. It is a top-down view of the HIPPI-MI, written from the viewpoint of an HIPPI-MI user. The conceptual relationship of these components is shown in figure 3.

The *service user* resides on the source system and is the agency that initiates HIPPI-MI transactions. The *service provider* collectively resides on both the source and target systems; its function is to take the service user's requests, relay and validate those requests on the target system, and present the validated transactions to the *target memory subsystem*.

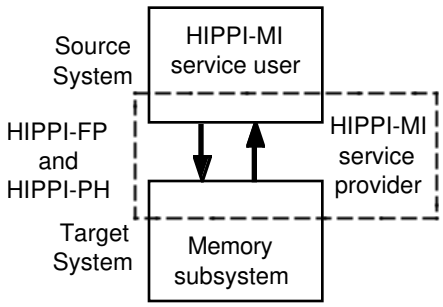

Figure 3 - Service primitive components

5.1.1 Types of primitives

HIPPI-MI service primitives are of four types.

- *Request* primitives are issued by the service user or target memory subsystem to initiate a service from the service provider. In this standard, a second Request primitive of the same name shall not be issued until the Confirm for the first request is received.

- *Confirm* primitives are issued by a service provider to acknowledge a Request.

- *Indicate* primitives are issued by the service provider to notify the service user or target memory subsystem of a local event. This primitive is similar in nature to an unsolicited interrupt. Note that the local event may have been caused by a service Request. In this standard, a second Indicate primitive of the same name shall not be issued until the Response for the first Indicate is received.

- *Response* primitives are issued by a service provider to acknowledge an Indicate.

5.1.2 Sequences of primitives

The order of execution of service primitives is not arbitrary. Logical and time sequence relationships exist for all described service primitives. Other valid sequences may exist. The sequence of events between peer users across the user/provider interface is illustrated in figure 4 and 5. HIPPI-MI primitives for initialize, read, write, and lock follow the sequence as shown in figure 4; substitute READ, WRITE, LOCK, and INIT, for "xxx" in figure 4 for the specific sequence. The primitives for message sequences are shown in figure 5.

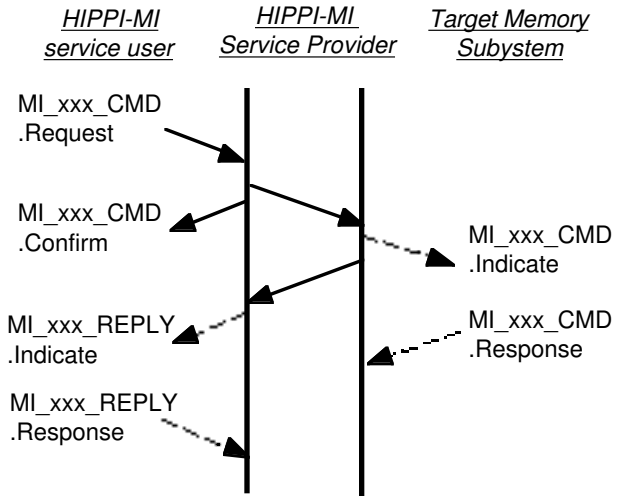

Figure 4 - Read, write, lock, and initialize service primitive sequences

The sequence of events is shown from top to bottom in figure 4 for the read, write, lock and initialize primitives. In general the operations include:

a) The source system service user requests a service via a MI_xxx_CMD.Request primitive, and passes the necessary parameters to the source system HIPPI-MI;

b) The source system HIPPI-MI shall verify that the target system supports the requested operation by checking the T_Parameters in the initialize reply packet. If supported, then the source system HIPPI-MI shall build a command packet with the parameters supplied by the service user, and appropriate local parameters, e.g., Access_Key from initialization reply packet, and shall append any appropriate data. The source system HIPPI-MI shall then forward this packet to the source system HIPPI-FP for delivery to the target system;

c) The source system HIPPI-MI shall issue a MI_xxx_CMD.Confirm primitive to the source system service user indicating if the operation was supported and the command packet sent to the target. This is an interlock; the service user shall not issue another

MI_xxx_CMD.Request until it has received the MI_xxx_CMD.Confirm;

d) The HIPPI-MI in the target system receives the command packet and checks it for validity and a proper Access_Key. If OK, it shall then execute the requested action, e.g., read. If a reply is requested for a read command via the Answer parameter, and for all other command packets, then the target system HIPPI-MI shall build the reply packet and forward it to the target system HIPPI-FP for delivery to the source system;

e) If the source system service user had requested that the target be signaled at the completion of the operation via the Signal bit, then the target system shall issue a MI_xxx_CMD.Indicate primitive. The target system HIPPI-MI shall not issue another MI_xxx_CMD.Indicate until it has received an MI_xxx_CMD.Response interlock primitive;

f) Upon receipt of the reply packet, the source system HIPPI-MI shall pass the results to the service user by issuing a MI_xxx_REPLY.Indicate primitive if Signal = 1. The source system HIPPI-MI shall not issue another MI_xxx_REPLY.Indicate until it receives a MI_xxx_REPLY.Response interlock primitive.

5.2 Primitives and packet relationship

Command packets are originated by the source system, and are associated with the MI_xxx_CMD primitives. Reply packets are originated by the target system, and are associated with the MI_xxx_REPLY primitives.

5.3 HIPPI-MI service primitives summary

The MI_xxx_CMD primitives use all four types, i.e., .Request, .Confirm, .Indicate, and .Response. The MI_xxx_REPLY primitives may use all four types, or only the .Indicate and .Response types as shown below. The primitives that are unique for each function (.Request and .Indicate) are detailed in 5.4 - 5.8. The handshake primitives (.Confirm and .Response) are detailed in 5.9.

The basic primitives are summarized below.

Initialization operations: (See 5.4)
MI_INIT_CMD (all four types)
MI_INIT_REPLY (.Indicate and .Response only)

Read data and cancel operations: (See 5.5)
MI_READ_CMD (all four types)
MI_READ_REPLY (.Indicate and .Response only)

Write data and No_Operation operations: (See 5.6)
MI_WRITE_CMD (all four types)
MI_WRITE_REPLY (.Indicate and .Response only)

Lock operations: (See 5.7)
MI_LOCK_CMD (all four types)
MI_LOCK_REPLY (.Indicate and .Response only)

Message operations: (See 5.8)
MI_MSG_CMD (all four types)
MI_MSG_REPLY (all four types)

5.4 Initialize primitives

Initialize primitives may be used to validate the source system user and inform the target system of source system limitations. See clause 7, and figures 6 and 7, for packet details. The initialize primitives, as shown in figure 4, also include the handshake primitives of 5.9 with xxx = INIT.

5.4.1 MI_INIT_CMD.Request

This primitive may be issued by the HIPPI-MI service user on the source system to exchange parameters with the target system.

Semantics **-** MI_INIT_CMD.Request (
S_id_1,2,3,
Signal,
Interrupt,
Transfer_Length,
User_Key)

S_id_1, S_id_2, and S_id_3 identify this request to the source system HIPPI-MI service user. (See 4.3)

Signal indicates if the target system is to be notified at the completion of this command. (See 4.4)

Interrupt indicates if the source system is to be notified when it receives the initialization reply packet. (See 4.3)

Transfer_Length specifies the number of bytes in the User_Key.

User_Key contains the value of the User_Key. (See 4.2)

Issued - This primitive is issued by the HIPPI-MI service user on the source system to exchange parameters with the target system.

Effect -

a) The source system HIPPI-MI service provider shall build the HIPPI-MI initialization command packet based upon the parameters supplied by the service user, and shall include the appropriate S_Parameters. It shall then forward the packet to the HIPPI-FP for delivery to the target system using the appropriate CCI.

b) The source system HIPPI-MI shall issue an MI_INIT_CMD.Confirm primitive to the service user.

c) When the target system HIPPI-MI receives the initialization command packet it shall check it for validity. If OK, it shall record the S_id_1,2,3 parameters, CCI, and S_Parameters for future use, and shall assign an Access_Key.

d) The target system HIPPI-MI shall build the initialization reply packet, including the Access_Key and appropriate T_Parameters, and forward it to the HIPPI-FP for delivery to the source system.

e) If Signal = 1 in the initialization command packet, then the target system HIPPI-MI shall issue a MI_INIT_CMD.Indicate primitive.

f) When the source system HIPPI-MI receives an initialization reply packet it shall check it for validity. If OK, it shall record the S_id_1,2,3 parameters, Access_Key, and T_Parameters for future use.

g) If Signal = 1 in the initialization reply packet, then the source system shall issue a MI_INIT_REPLY.Indicate primitive to the source system service user.

5.4.2 MI_INIT_REPLY.Indicate

This primitive is use to pass the information in an initialization reply packet to the source system service user.

Semantics - MI_INIT_REPLY.Request (
S_id_1,2,3,
Eight,
Four,
Message,
Write,
Read,
T_Granularity,
MI_Status,
Status)

S_id_1, S_id_2, and S_id_3 identify this initialization reply packet to the source system service user. (See 4.3)

Eight indicates if the target system supports 8-byte Lockwords. (See 7.3)

Four indicates if the target system supports 4-byte Lockwords. (See 7.3)

Message indicates if the target system supports message operations. (See 7.3)

Write indicates if the target system supports write operations. (See 7.3)

Read indicates if the target system supports read operations. (See 7.3)

T_Granularity specifies the granularity of transfers from the source system to the target system. (See 7.3)

MI_Status indicates if the operation was successful and, if not, then the reason for the failure. (See 7.1)

Status indicates if the initialization reply packet was received correctly, e.g., parity or LLRC errors detected.

Issued - The source system HIPPI-MI service provider shall issue this primitive to the HIPPI-MI service user upon receiving a packet with Operation = Initialize reply and Signal = 1.

Effect - This completes an initialization operation. If the MI_Status indicates that the initialization sequence completed successfully the the HIPPI-MI service user may issue commands that are supported by the target system, using the Access_Key supplied.

5.4.3 MI_INIT_CMD.Indicate

This primitive signals the target memory subsystem that the source system service user had requested that the target memory subsystem be signaled at the completion of this initialization operation.

Semantics - MI_INIT_CMD.Indicate (
CCI,
S_id_1,2,3,
Interrupt,
S_Parameters,
Transfer_Length,
User_Key,
Status)

CCI contains the routing information used to deliver the packet. (See 4.12)

Status indicates if the initialization command packet was received correctly, e.g., parity or LLRC errors detected.

The other parameters contain the same values as received in the initialization command packet. (See 5.4.1)

Issued - The target system HIPPI-MI shall issue this primitive when it completes the initialization operation.

Effect - Unspecified

5.5 Read data primitives

Read data primitives are used to transfer data from the target system's memory subsystem to the source system. See clause 7, and figures 8 and 9, for packet details. The read data primitives, as shown in figure 4, also include the handshake primitives of 5.9 with xxx = READ. Read data primitives are also used for cancel sequences (see 4.10).

5.5.1 MI_READ_CMD.Request

This primitive is issued by the HIPPI-MI service user to request a memory read from the target system.

Semantics - MI_READ_CMD.Request (
S_id_1,2,3,
T_Buffer_id,
T_First_Byte_Offset,
Transfer_Length,
Signal,
Interrupt)

S_id_1, S_id_2, and S_id_3 identify this request to the source system HIPPI-MI service user. (See 4.3)

T_Buffer_id identifies the buffer to be read. (See 4.4)

T_First_Byte_Offset specifies the relative location for the first byte of the read operation. (See 4.4)

Transfer_Length specifies the number of bytes to be read. (See 4.6)

Signal indicates if the target system is to be notified at the completion of this command. (See 4.4)

Interrupt indicates if the source system is to be notified when it receives the read reply packet. (See 4.3)

Issued - The HIPPI-MI service user issues this primitive to request a memory read on the target system.

Effect -

a) The source system HIPPI-MI service provider shall verify that the target system supports read operations. If so, it shall build the HIPPI-MI read command packet based upon the parameters supplied by the service user, and shall include the appropriate Access_Key. It shall then forward the packet to the HIPPI-FP for delivery to the target system using the appropriate CCI.

b) The source system HIPPI-MI shall issue an MI_READ_CMD.Confirm primitive to the service user.

c) When the target system HIPPI-MI receives the read command packet it shall check it for validity and a proper Access_Key. If OK, it shall read the requested data from the target memory subsystem.

d) The target system HIPPI-MI shall build the read reply packet and forward it to the HIPPI-FP for delivery to the source system.

e) If Signal = 1 in the read command packet, then the target system HIPPI-MI shall issue a MI_READ_CMD.Indicate primitive.

f) If Signal = 1 in the read reply packet, then the source system shall issue a MI_READ_REPLY.Indicate primitive to the source system service user.

5.5.2 MI_READ_REPLY.Indicate

This primitive is use to pass the information in a read reply packet to the source system service user.

Semantics **-** MI_READ_REPLY.Indicate (
S_id_1,2,3,
T_Buffer_id,
T_First_Byte_Offset,
Transfer_Length,
MI_Status,
Data
Status)

S_id_1, S_id_2, and S_id_3 identify this read reply packet to the source system service user. (See 4.3)

Transfer_Length indicates the number of bytes actually read from the target memory subsystem. Note that this may not be the same as the number of bytes requested by the service user in the MI_READ_CMD.Request.

MI_Status indicates success, or nature of failure. (See 7.1)

Data contains the data read from the memory subsystem.

Status indicates if the read reply packet was received correctly, e.g., parity or LLRC errors detected.

Issued - The source system HIPPI-MI service provider shall issue this primitive to the HIPPI-MI service user upon receiving a packet with Operation = Read reply and Signal = 1.

Effect - This completes a HIPPI-MI read operation.

5.5.3 MI_READ_CMD.Indicate

This primitive signals the target memory subsystem that the source system service user had requested that the target memory subsystem be signaled at the completion of this read operation.

Semantics **-** MI_READ_CMD.Indicate (
CCI,
S_id_1,2,3,
Access_Key,
T_Buffer_id,
T_First_Byte_Offset,
Transfer_Length,
Interrupt
Status)

CCI contains the routing information used to deliver the packet. (See 4.12)

Status indicates if the read command packet was received correctly, e.g., parity or LLRC errors detected.

The other parameters contain the same values as received in the read command packet. (See 5.5.1)

Issued - The target system HIPPI-MI service provider shall issue this primitive at the completion of the command if Signal = 1.

Effect - Unspecified

5.6 Write data primitives

Write data primitives are used to transfer data from the source system to the target system memory subsystem. See clause 7, and figures 10 and 11, for packet details. The write data primitives, as shown in figure 4, also include the handshake primitives of 5.9 with xxx = WRITE. Write data primitives are also used for No_Operation sequences (see 4.11).

5.6.1 MI_WRITE_CMD.Request

This primitive is issued by the HIPPI-MI service user on the source system to write to a buffer on the target system.

Semantics **-** MI_WRITE_CMD.Request (
S_id_1,2,3,
T_Buffer_id,
T_First_Byte_Offset,
Transfer_Length,
Signal,
Interrupt,
Answer,
Data)

S_id_1, S_id_2, and S_id_3 identify this request to the source system HIPPI-MI service user. (See 4.3)

T_Buffer_id identifies the beginning of the buffer to be written. (See 4.4)

T_First_Byte_Offset specifies the relative location for the first byte of the write operation. (See 4.4)

Transfer_Length specifies the number of bytes to be written. The source system may specify a larger number than will actually be transferred. (See 4.7)

Signal indicates if the target system is to be notified at the completion of this command. (See 4.4)

Interrupt indicates if the source system is to be notified when it receives the write reply packet. (See 4.3)

Answer indicates if the target system is to generate a reply packet in response to this command packet. (See 4.7)

Data contains the data to be written into the memory of the target system. (See 4.7)

Issued - This primitive is issued by the HIPPI-MI service user on the source system to write to a buffer on the target system.

Effect -

a) The source system HIPPI-MI service provider shall verify that the target system supports write operations. If so, it shall build the HIPPI-MI write command packet based upon the parameters supplied by the service user, and shall include the appropriate Access_Key. It shall then forward the packet to the HIPPI-FP for delivery to the target system using the appropriate CCI.

b) The source system HIPPI-MI shall issue an MI_WRITE_CMD.Confirm primitive to the service user.

c) When the target system HIPPI-MI receives the write command packet it shall check it for validity and a proper Access_Key. If OK, it shall write the requested data to the target memory subsystem.

d) If Answer = 1 in the write command packet, then the target system HIPPI-MI shall build the write reply packet and forward it to the HIPPI-FP for delivery to the source system.

e) If Signal = 1 in the write command packet, then the target system HIPPI-MI shall issue a MI_WRITE_CMD .Indicate primitive.

f) If Signal = 1 in the write reply packet, then the source system shall issue a MI_WRITE_REPLY .Indicate primitive to the source system service user.

5.6.2 MI_WRITE_REPLY.Indicate

This primitive is use to pass the information in a write reply packet to the source system service user.

Semantics **-** MI_WRITE_REPLY.Indicate (
S_id_1,2,3,
T_Buffer_id,
T_First_Byte_Offset,
Transfer_Length,
MI_Status,
Status)

S_id_1, S_id_2, and S_id_3 identify this write reply packet to the source system service user. (See 4.3)

Transfer_Length indicates the number of bytes actually written to the target memory subsystem. Note that this may not be the same as the number of bytes requested by the service user in the MI_WRITE_CMD.Request.

MI_Status indicates success, or nature of failure. (See 7.1)

Status indicates if the write reply packet was received correctly, e.g., parity or LLRC errors detected.

Issued - The source system HIPPI-MI service provider shall issue this primitive to the HIPPI-MI service user upon receiving a packet with Operation = Write reply and Signal = 1.

Effect - This completes a HIPPI-MI write operation.

5.6.3 MI_WRITE_CMD.Indicate

This primitive signals the target memory subsystem that the source system service user had requested that the target memory subsystem be signaled at the completion of this write operation.

Semantics **-** MI_WRITE_CMD.Indicate (
CCI,
S_id_1,2,3,
Access_Key,
T_Buffer_id,
T_First_Byte_Offset,
Transfer_Length,
Interrupt
Answer,
Data,
Status)

CCI contains the routing information used to deliver the packet. (See 4.12)

Status indicates if the write command packet was received correctly, e.g., parity or LLRC errors detected.

The other parameters contain the same values as received in the write command packet. (See 5.6.1)

Issued - The target system HIPPI-MI service provider shall issue this primitive at the completion of the command if Signal = 1.

Effect - Unspecified

5.7 Lock primitives

Lock primitives are used by the source system to manipulate Lockwords in the target system. See clause 7, and figures 12 and 13, for packet details. The lock primitives, as shown in figure 4, also include the handshake primitives of 5.9 with xxx = LOCK.

5.7.1 MI_LOCK_CMD.Request

This primitive is issued by the HIPPI-MI service user to request a memory lock on the target system.

Semantics - MI_LOCK_CMD.Request (
S_id_1,2,3,
T_Buffer_id,
T_First_Byte_Offset,
Transfer_Length,
Signal,
Interrupt,
Operation,
Check_Value,
New_Data)

S_id_1, S_id_2, and S_id_3 identify this request to the source system HIPPI-MI service user. (See 4.3)

T_Buffer_id identifies the beginning of the buffer containing the Lockword. (See 4.4)

T_First_Byte_Offset specifies the relative location for the first byte of the Lockword. (See 4.4)

Transfer_Length specifies the number of bytes (8 or 16) in the Lockword pair of Check_Value and New_Data. (See 4.8)

Signal indicates if the target system is to be notified at the completion of this command. (See 4.4)

Interrupt indicates if the source system is to be notified when it receives the lock reply packet. (See 4.3)

Operation specifies the type of lock operation to be performed. (See 4.8 and 7.1)

Check_Value is compared with the original contents of the Lockword during the operation. (See 4.8)

New_Data specifies the value used to modify the Lockword in the target system memory if the lock operation is successful. (See 4.8)

Issued - The HIPPI-MI service user issues this primitive to request a lock operation on the target system.

Effect -

a) The source system HIPPI-MI service provider shall verify that the target system supports the requested Lockword size. If so, it shall build the HIPPI-MI lock command packet based upon the parameters supplied by the service user, and shall include the appropriate Access_Key. It shall then forward the packet to the HIPPI-FP for delivery to the target system using the appropriate CCI.

b) The source system HIPPI-MI shall issue an MI_LOCK_CMD.Confirm primitive to the service user.

c) When the target system HIPPI-MI receives the lock command packet it shall check it for validity and a proper Access_Key. If OK, it shall perform the requested lock operation on the target memory subsystem.

d) The target system HIPPI-MI shall build the lock reply packet and forward it to the HIPPI-FP for delivery to the source system.

e) If Signal = 1 in the lock command packet, then the target system HIPPI-MI shall issue a MI_LOCK_CMD .Indicate primitive.

f) If Signal = 1 in the read reply packet, then the source system shall issue a MI_LOCK_REPLY .Indicate primitive to the source system service user.

5.7.2 MI_LOCK_REPLY.Indicate

This primitive is use to pass the information in a lock reply packet to the source system service user.

Semantics - MI_LOCK_REPLY.Indicate (
S_id_1,2,3,
T_Buffer_id,
T_First_Byte_Offset,
Transfer_Length,
Operation,
Test,
Check_Value,
Old_Data,
MI_Status,
Status)

S_id_1, S_id_2, and S_id_3 identify this lock reply packet to the source system service user. (See 4.3)

T_Buffer_id, T_First_Byte_Offset, Transfer_Length, Operation, and Check_Value, contain the same values as in the lock command packet. (See 5.7.1)

Test is an indication of whether the lock test was successful. (See 4.8)

Old_Data contains the contents of the memory location (Lockword) in the target system before the lock operation took place. (See 4.8)

MI_Status indicates success, or nature of failure. (See 7.1)

Status indicates if the write reply packet was received correctly, e.g., parity or LLRC errors detected.

Issued - The source system HIPPI-MI service provider shall issue this primitive to the HIPPI-MI service user upon receiving a packet with Operation = Lock... reply and Signal = 1.

Effect - This completes a HIPPI-MI lock operation.

5.7.3 MI_LOCK_CMD.Indicate

This primitive signals the target memory subsystem that the source system service user had requested that the target memory subsystem be signaled at the completion of this lock operation.

Semantics - MI_LOCK_CMD.Indicate (
CCI,
S_id_1,2,3,
Access_Key,
T_Buffer_id,
T_First_Byte_Offset,
Transfer_Length,
Signal,
Interrupt,
Operation
Check_Value,
New_Data,
Status)

CCI contains the routing information used to deliver the packet. (See 4.12)

Status indicates if the lock command packet was received correctly, e.g., parity or LLRC errors detected.

The other parameters contain the same values as received in the lock command packet. (See 5.7.1)

Issued - The target system HIPPI-MI service provider shall issue this primitive at the completion of the command if Signal = 1.

Effect - Unspecified

5.8 Message primitives

Message primitives are used to transfer commands and parameters between the source and target systems. See clause 7, and figures 14 and 15, for packet details. The message primitives, as shown in figure 5, also include the handshake primitives of 5.9 with xxx = MSG.

5.8.1 MI_MSG_CMD.Request

This primitive is issued by the HIPPI-MI service user to send a message to an object in the target system.

Semantics - MI_MSG_CMD.Request (
S_id_1,2,3,
Object_id,
Transfer_Length,
Signal,
Interrupt,
Message_Operation,
Message_Data)

S_id_1, S_id_2, and S_id_3 identify this request to the source system HIPPI-MI service user. (See 4.3)

Object_id identifies the specific object in the target system. (See 4.9)

Transfer_Length specifies the number of bytes of Message_Data. (See 4.6)

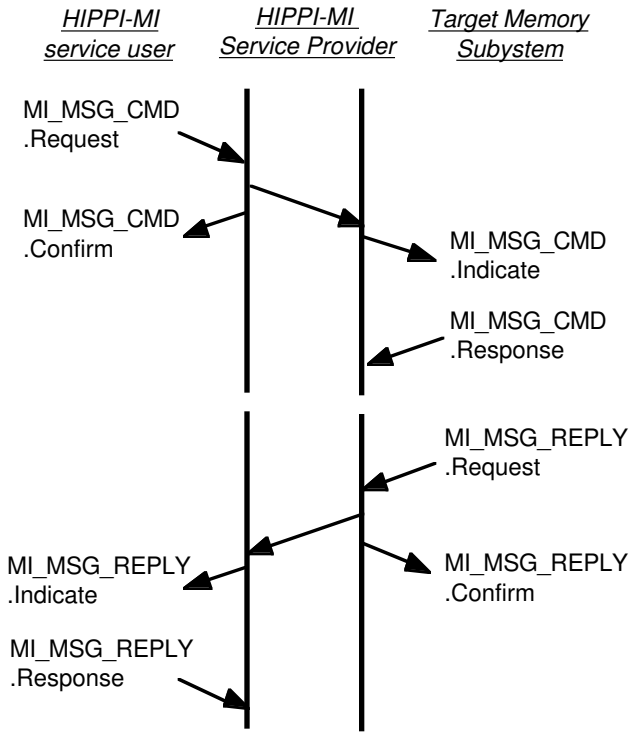

Figure 5 - Message service primitive sequence

Signal indicates if the target system is to be notified at the completion of this command. (See 4.4)

Interrupt indicates if the source system is to be notified when it receives the message reply packet. (See 4.3)

Message_Operation defines the operation to be performed. (See 7.6)

Message_Data contains the parameters, if any, for the requested operation.

Issued - The HIPPI-MI service user issues this primitive to pass a message to an object in the target system.

Effect -

a) The source system HIPPI-MI service provider shall verify that the target system supports message operations. If so, it shall build the HIPPI-MI message command packet based upon the parameters supplied by the service user, and shall include the appropriate Access_Key. It shall then forward the packet to the HIPPI-FP for delivery to the target system using the appropriate CCI.

b) The source system HIPPI-MI shall issue an MI_MSG_CMD.Confirm primitive to the service user.

c) When the target system HIPPI-MI receives the message command packet it shall check it for validity and a proper Access_Key. If OK, and Signal = 1, then it shall issue a MI_MSG_CMD.Indicate primitive.

5.8.2 MI_MSG_CMD.Indicate

This primitive is use to pass the information in a message command packet to the target system object.

Semantics - MI_MSG_CMD.Indicate (
CCI,
S_id_1,2,3,
Object_id,
Transfer_Length,
Interrupt,
Message_Operation,
Message_Data,
Status)

CCI contains the routing information used to deliver the packet. (See 4.12)

S_id_1, S_id_2, and S_id_3 identify this message reply packet to the source system service user. (See 4.3)

Status indicates if the message command packet was received correctly, e.g., parity or LLRC errors detected.

The other parameters contain the same values as received in the message command packet. (See 5.8.1)

Issued - The target system HIPPI-MI object shall issue this primitive after the MI header parameters of a lock command packet have been validated and Signal = 1.

Effect - The target system object shall perform the requested operation using the parameters supplied in this primitive.

5.8.3 MI_MSG_REPLY.Request

The target system uses this primitive to return the results of a message command to the source system.

Semantics - MI_MSG_REPLY.Request (
S_id_1,2,3,
Object_id,
Transfer_Length,
Signal,
Interrupt,
Message_Operation,
Message_Reply_Data,
MI_Status)

Transfer_Length specifies the number of bytes of Message_Reply_Data. (See 4.6)

Signal indicates if the source system is to be notified at the completion of this command and contains the same value as the Interrupt parameter received in the message command packet. (See 4.4)

MI_Status indicates success, or nature of failure. (See 7.1)

Message_Reply_Data contains the parameters, if any, resulting from the operation performed by the object in the target system.

S_id_1,2,3, Object_id, and Message_Operation are the same as received in the message command packet. (See 5.8.1)

Issued - The target system object shall issue this primitive, after completing the operation requested in the message command packet, to pass the results of the operation to the source system.

Effect -

a) The target system HIPPI-MI shall build the message reply packet based upon the parameters supplied, and shall include the appropriate Access_Key. It shall then forward the packet to the HIPPI-FP for delivery to the source system using the appropriate CCI.

b) The target system HIPPI-MI shall issue an MI_MSG_REPLY.Confirm primitive to the service user.

c) When the source system HIPPI-MI receives the message reply packet it shall check it for validity. If OK, and Signal = 1, then it shall issue a MI_MSG_REPLY.Indicate primitive.

5.8.4 MI_MSG_REPLY.Indicate

This primitive is use to pass the information in a message reply packet to the source system service user.

Semantics - MI_MSG_REPLY.Indicate (
S_id_1,2,3,
Object_id,
Transfer_Length,
Message_Operation,
Message_Reply_Data
MI_Status,
Status)

S_id_1,2,3, Object_id, Transfer_Length, Message _Operation, Message_Reply_Data, and MI_Status, contain the same values as received in the message reply packet. (See 5.8.3)

Status indicates if the message reply packet was received correctly, e.g., parity or LLRC errors detected.

Issued - The source system HIPPI-MI service provider shall issue this primitive to the HIPPI-MI service user upon receiving a message reply packet.

Effect - This completes a HIPPI-MI message operation.

5.9 Handshake primitives

The .Confirm and .Response primitives are interspersed with the other primitive types, as shown in figures 4 and 5, providing handshake functions. The .Confirm and .Response primitive types are used in a consistent fashion in the read, write, lock, message, and initialize sequences. This subclause defines these primitives in a generic manner, with xxx substituting for READ, WRITE, LOCK, MSG, and INIT.

5.9.1 MI_xxx_CMD.Confirm

This primitive acknowledges the MI_xxx_CMD.Request from the HIPPI-MI service user on the source system.

Semantics **-** MI_xxx_CMD.Confirm (Status)

Status shall be one of the following:

- Accept - The HIPPI-FP has completed the connection to the target system and accepted the command packet for transmission.

- Reject - The Target system does not support the requested operation, or has rejected the connection request. The command packet was not transmitted.

- Timeout - The Target system did not respond to the HIPPI-FP connection request within the timeout period. The command packet was not transmitted.

Issued - The source system HIPPI-MI service provider shall issue this primitive to the HIPPI-MI service user issuing an MI_xxx_CMD.Request.

Effect - Unspecified.

5.9.2 MI_xxx_CMD.Response

This primitive acknowledges that the target memory subsystem has received an MI_xxx_CMD.Indicate.

Semantics **-** MI_xxx_CMD.Response

Issued - The target memory subsystem issues this primitive to acknowledge receipt of the MI_xxx_CMD .Indicate.

Effect **-** The target system HIPPI-MI service provider is enabled to issue another MI_xxx_CMD.Indicate.

5.9.3 MI_xxx_REPLY.Confirm

This primitive acknowledges the MI_xxx_REPLY.Request from the target memory subsystem on the target system.

Semantics **-** MI_xxx_REPLY.Confirm (Status)

Status shall be one of the following:

- Accept - The target system HIPPI-FP has completed the connection to the source system and accepted the reply packet for transmission.

- Reject - The source system has rejected the connection request. The reply packet was not transmitted.

- Timeout - The Source system did not respond to the HIPPI-FP connection request within the timeout period. The reply packet was not transmitted.

Issued - The target system HIPPI-MI service provider shall issue this primitive to the target memory subsystem issuing an MI_xxx_REPLY.Request.

Effect - Unspecified.

5.9.4 MI_xxx_REPLY.Response

This primitive acknowledges that the HIPPI-MI service user has received an MI_xxx_REPLY.Indicate.

Semantics - MI_xxx_REPLY.Response

Issued - The source system HIPPI-MI service user issues this primitive to acknowledge the receipt of an MI_xxx_REPLY. Indicate.

Effect - The source system HIPPI-MI service provider is enabled to issue another MI_xxx_REPLY.Indicate.

6 HIPPI-FP interface to HIPPI-MI

A summary of the HIPPI-FP primitives used to connect the HIPPI-MI to the HIPPI-FP is listed below. The complete specifications of the primitives are contained in the HIPPI Framing Protocol (HIPPI-FP) document.

FP_TRANSFER.Request (
CCI,
ULP-id,
D1_Size,
D1_Data_Set,
D2_Size,
D2_Data_Set,
Keep_Connection,
Start_D2_on_Burst_Boundary)

FP_TRANSFER.Confirm (Status)

FP_TRANSFER_D1.Indicate (
ULP-id,
CCI,
Status,
D2_Size,
D2_Offset,
D1_Area_Size,
D1_Area)

FP_TRANSFER_D2.Indicate (
ULP-id,
CCI,
Status,
D2_Size,
D2_Offset,
D2_Data_Set)

FP_TRANSFER.Response

HIPPI-MI is a user of the HIPPI-FP. The HIPPI-MI specification utilizes the following values for the parameters of the HIPPI-FP service interface.

- ULP-id = 2 (for read, write, and Lock)
- ULP-id = 3 (for initialization, cancel, and message)
- D1 Present = 1.
- D1_Area_Size $\geq$ 4.
- D1_Data_Set size = 32 bytes
- B bit (start D2_Area on burst boundary) may or may not be set.
- D2_Offset field may be non-zero although HIPPI-MI implementations are not required to support it.

7 HIPPI-MI packet formats

7.1 Common fields

The Signal, Interrupt, Operation, MI_Status, S_id_1,2,3, Transfer_Length, and Access_Key fields are common to all types of the packets. The locations of these fields are shown in figures 6 - 15.

S = Signal (bit 31) indicates whether a higher level process in the destination shall be signaled at the completion of the operation, such as through the generation of an interrupt.

I = Interrupt (bit 30) indicates whether a higher level process in the source system shall be signaled at the completion of the operation, such as through the generation of an interrupt. The value of Interrupt bit is copied into the Signal bit of a reply packet by the target system.

Operation (bits 29-24) specifies the function of the packet. The operation codes for ULP-id = 2 are:

000000 - No_Operation command
000001 - No_Operation reply
000010 - Reserved
000011 - Reserved
000100 - Read command
000101 - Read reply
000110 - Write command
000111 - Write reply
001000 - Lock Swap command
001001 - Lock Swap reply
001010 - Lock Compare and Swap command
001011 - Lock Compare and Swap reply
001100 - Lock Fetch and Add command
001101 - Lock Fetch and Add reply
001110 - Reserved for lock operation
001111 - Reserved for lock operation
1xxxxx - Vendor unique
- other values are reserved

The operation codes for ULP-id = 3 are:

000000 - No_Operation command
000001 - No_Operation reply
000010 - Initialize command
000011 - Initialize reply
000100 - Cancel command
000101 - Cancel reply
000110 - Message command
000111 - Message reply
1xxxxx - Vendor unique
- other values are reserved

MI_Status (3 bits) defines the status of the requested operation. A reply MI_Status value of all zeroes indicates successful completion of the operation. If errors are indicated, the other fields of the packet may be invalid. MI_Status is valid only in reply packets. The MI_Status field shall be set to all zeroes in command packets.

D = Access Denied (bit 18) - a 1 indicates that either the User_Key submitted as part of the initialization sequence or the Access_Key supplied with a command was not acceptable to the target system.

X = Memory Subsystem Error (bit 17) - a 1 indicates that an error was detected by the memory subsystem during execution of the operation.

N = Invalid Request (bit 16) - a 1 indicates that one or more parameters in the command packet were unacceptable to the target system.

Note: The primary purpose of the X bit is to signal the occurrence of a hardware-type error. Some implementations may also indicate certain errors by breaking the connection with an illegal end termination.

S_id_1 (bits 15-0), S_id_2 (32 bits), and S_id_3 (32 bits) contain user information that allows association between a command and a reply. These parameters are supplied by the source system user with each command, and shall be returned unmodified with the reply from the target system. (See 4.3)

Transfer_Length (32 bits) specifies the number of bytes transferred. Depending on the packet type this field has slightly different meanings; refer to the specific sequence descriptions in clause 4 for details. It should be noted that the Transfer_Length value is not necessarily equal to the D2_Size parameter in the HIPPI-FP header.

Access_Key (64 bits) grants permission to perform the operation. The Access_Key value must be the same as the Access_Key returned in the initialization reply. (See 4.2)

Reserved or Rsv - Throughout this document, all of the reserved bits shall be transmitted as zeros, but do not need to be checked upon reception.

7.2 Initialization command information fields

The following initialization command fields contain information about the HIPPI-MI implementation on the source system. (See 4.5) The initialization command packet format is shown in figure 6.

Transfer_Length (32 bits) specifies the number of bytes in the S_Parameters, T_Parameters, and User_Key fields. The minimum value for this field is 4; maximum is 68.

Access_Key, T_Buffer_id and T_First_Byte_Offset - zeros shall be transmitted, but not checked by the target system.

S_Parameters shall contain the source system parameters:

B = Start_D2_on_Burst (bit 25) - location of the read data in read reply packets from the target system:
0 = D2_Area may start in the first burst
1 = D2_Area shall start at the beginning of the second burst

Z = Zero (bit 24) - indicates if the source system requires zero D2_Offsets in reply packets:
0 = not required
1 = required

S_Granularity (bits 23-16) - the minimum Transfer_Length or T_First_Byte_Offset granularity that could be requested by the source system. Specified as 2^n bytes:
00000000 = 1 byte
00000001 = 2 bytes
00000010 = 4 bytes
00000011 = 8 bytes
00000100 = 16 bytes
00000101 = 32 bytes
etc. through 11111111

T_Parameters (bits 15-0) - zeros shall be transmitted, but not checked by the target system.

User_Key (0-64 bytes) - contains up to 64 bytes of information that identifies the source system. (See 4.2)

7.3 Initialization reply information fields

The following initialization reply fields contain information about the HIPPI-MI implementation on the target system. (See 4.5) The initialization reply packet format shown in figure 7.

Transfer_Length = 4 (the number of bytes in the S_Parameter and T_Parameter fields)

Access_Key (64 bits) - contains the Access_Key the source system shall use when issuing commands to the target system. (See 4.2)

T_Buffer_id and T_First_Byte_Offset - zeros shall be transmitted, but not checked by the source system.

S_Parameters (bits 31-16) - shall be the same as received in the initialization command packet. (See 7.2)

The T_Parameters shall contain the target system parameters:

E = Eight (bit 14) - indicates if 8-byte Lockwords are supported:
0 = not supported
1 = supported

F = Four (bit 13) - indicates if 4-byte Lockwords are supported:
0 = not supported
1 = supported

M = Message (bit 12) - indicates if message commands are supported:
0 = not supported
1 = supported

W = Write (bit 11) - indicates if write commands are supported:
0 = not supported
1 = supported

R = Read (bit 10) - indicates if read commands are supported:
0 = not supported
1 = supported

B = Start_D2_on_Burst (bit 9) - location of the write data in write command packets from the source system:
0 = D2_Area may start in the first burst
1 = D2_Area shall start at the beginning of the second burst

Z = Zero (bit 8) - indicates if the target system requires that zero D2_Offsets be used in commands packets from the source system:
0 = not required
1 = required

T_Granularity (bits 7-0) - the minimum Transfer_Length or T_First_Byte_Offset granularity supported by the target system. Specified as 2^n bytes:
00000000 = 1 byte
00000001 = 2 bytes
00000010 = 4 bytes
00000011 = 8 bytes
00000100 = 16 bytes
00000101 = 32 bytes
etc. through 11111111

7.4 Read, write, and lock parameter fields

The T_Buffer_id, and T_First_Byte_Offset, fields are common to read, write, and lock packets. The locations of these fields are shown in figures 8 through 13. The values for these fields are:

T_Buffer_id (32 bits) identifies the memory buffer in the target system that is to be accessed. (See 4.4)

T_First_Byte_Offset (32 bits) specifies the relative location of the first byte of the operation. (See 4.4)

A = Answer (bit 23) indicates if the target system shall return a reply to the source system for a write or No_Operation command.

7.5 Lock information fields

The locations of the lock command and reply fields are shown in figures 12 and 13. A discussion of the lock operations is contained in 4.8. The size, in bytes, of each of the Check_Value, Old_Data, and New_Data fields is four or eight and is one-half of the Transfer_Length value.

T = Test (bit 19) - is set to 1 if the Check_Value in a lock command was equal to the Old_Data originally at that memory location, and is set to 0 if it was unequal. T is valid only in lock reply packets. A valid Test parameter is returned for all lock operations whose MI_Status indicates Operation Successful; all other lock operations shall return T = 0.

Check_Value (4 or 8 bytes) - a value provided by the source system to compare against Old_Data.

Old_Data (4 or 8 bytes) - the value in the target system's Lockword before the lock operation begins. All lock operations return Old_Data to the source system in a lock reply packet.

New_Data (4 or 8 bytes) - the value provided by the source

system to replace or modify Old_Data.

7.6 Message information fields

A discussion of the message sequence is contained in 4.9. The Access_Key and Transfer_Length are the same as specified in 7.4. The message command and reply packets are shown in figures 14 and 15.

Object_id (32 bits) - identifies the specific object in the target system.

Message_Operation (32 bits) - identifies the operation to be performed by the target system object. All values of Message_Operation are vendor specific.

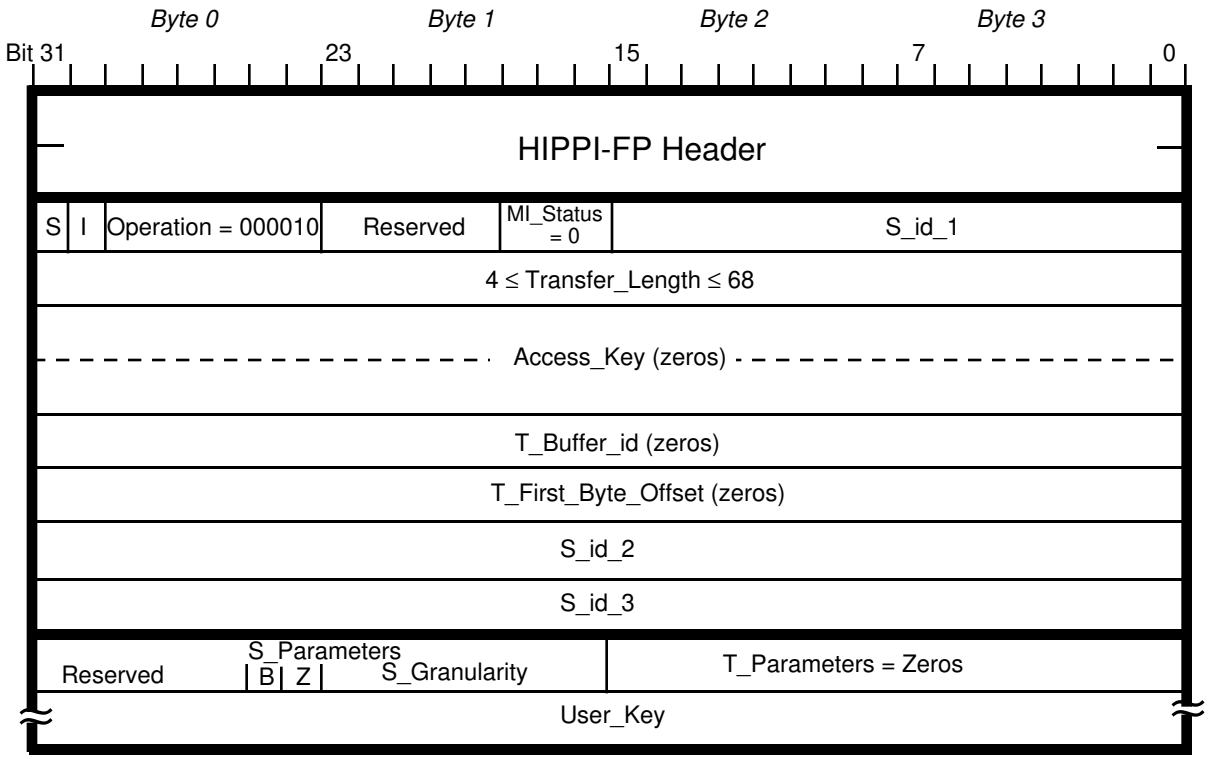

Figure 6 - HIPPI-MI initialization command packet format

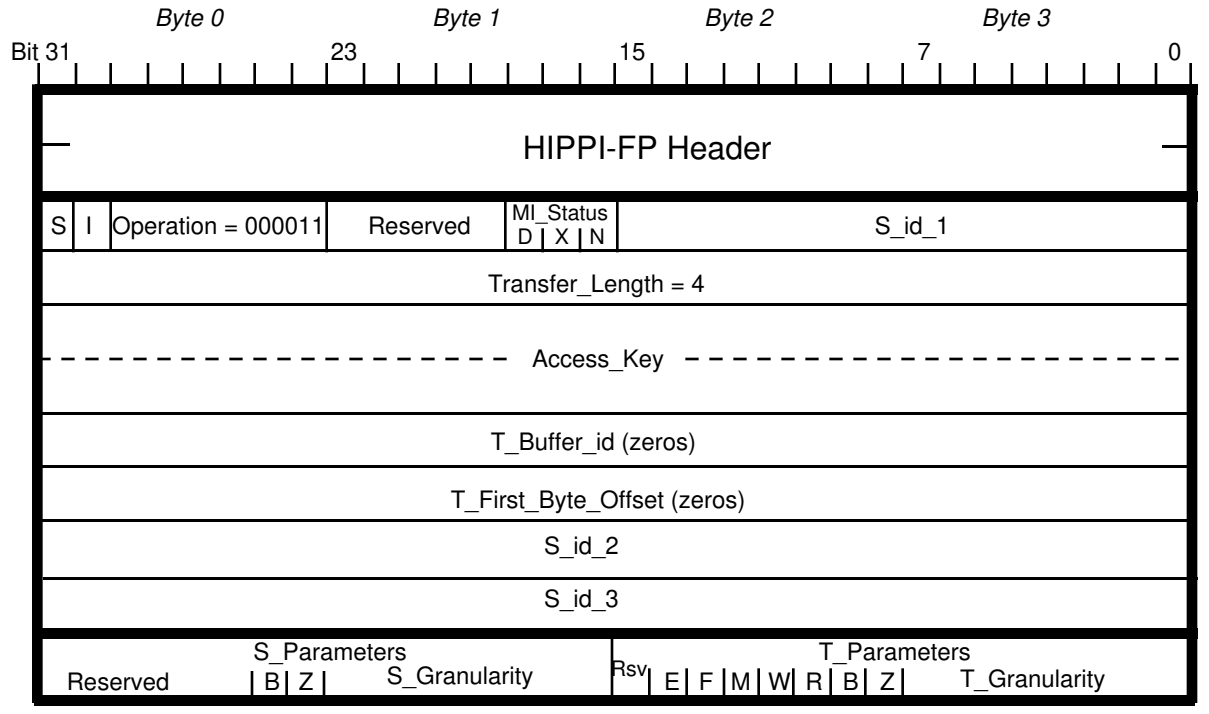

Figure 7 - HIPPI-MI initialization reply packet format

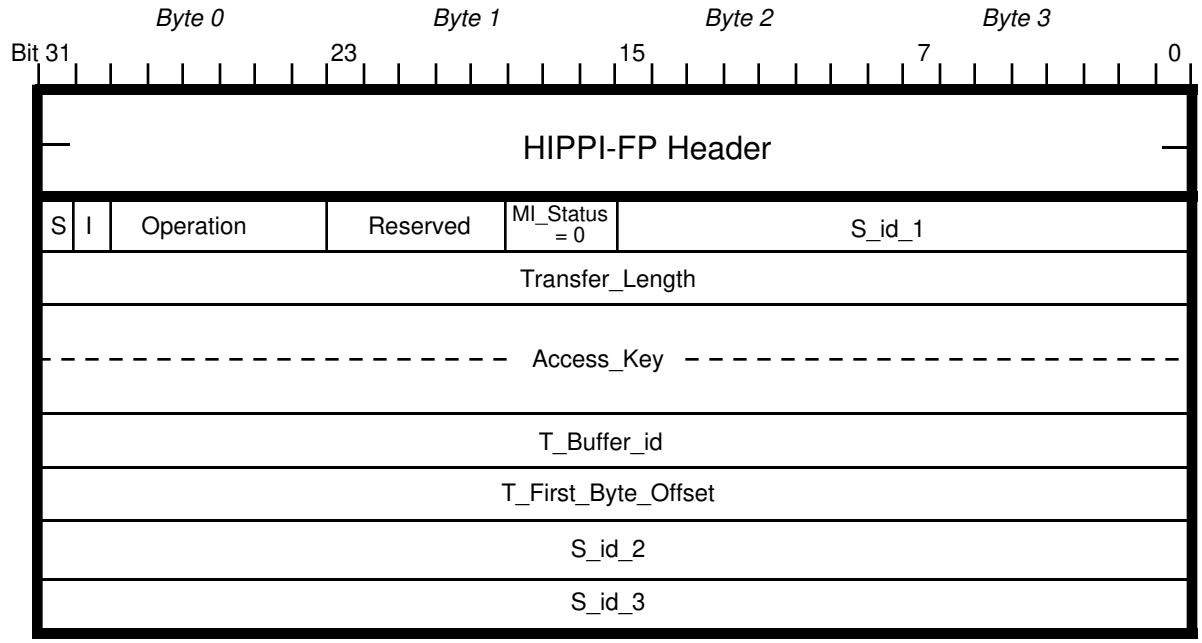

Figure 8 - HIPPI-MI read and cancel command packet formats

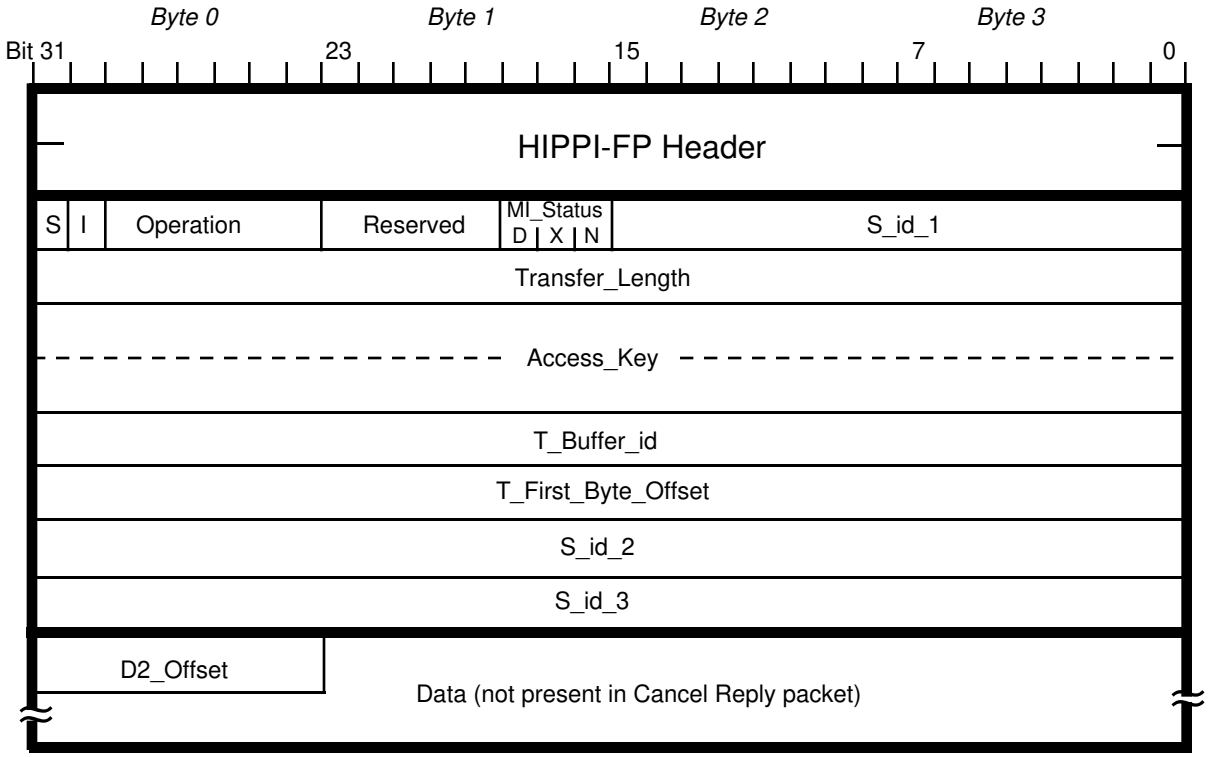

Figure 9 - HIPPI-MI read and cancel reply packet formats

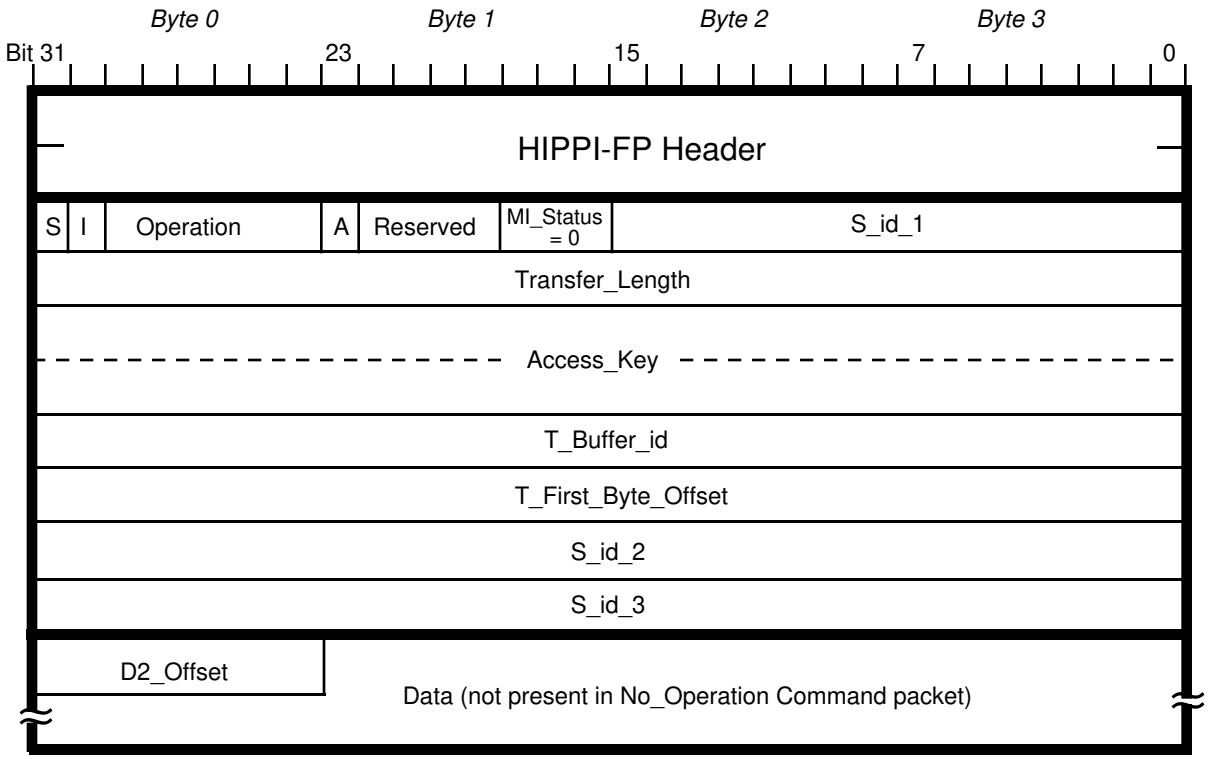

Figure 10 - HIPPI-MI write and No_Operation command packet formats

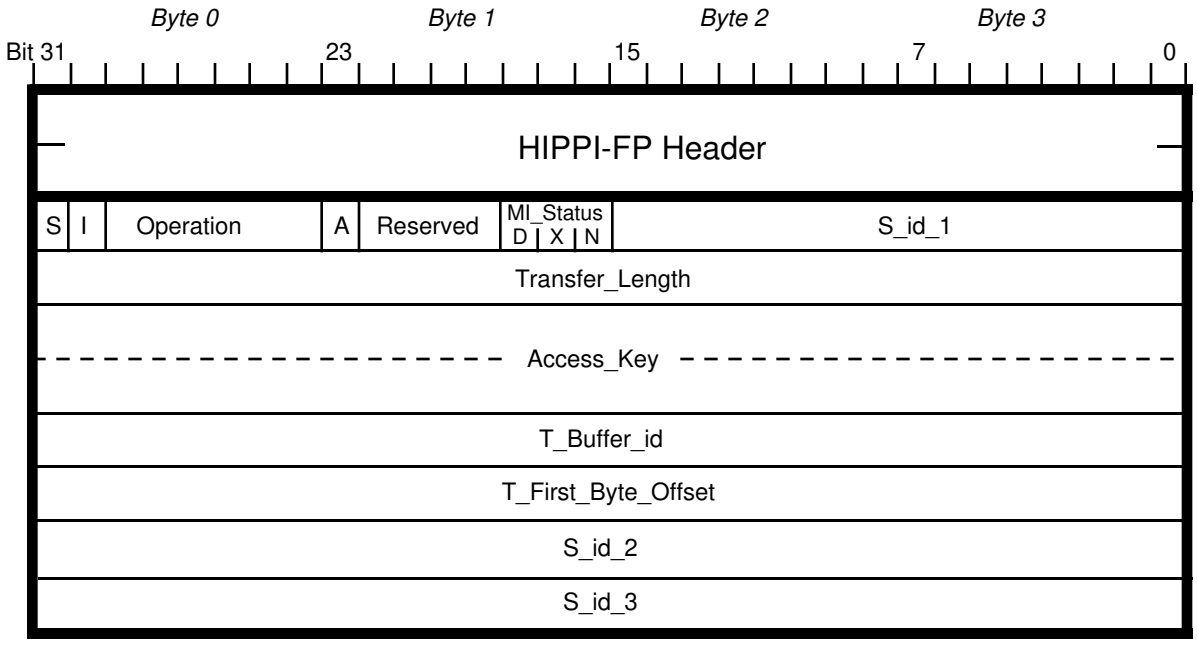

Figure 11 - HIPPI-MI write and No_Operation reply packet formats

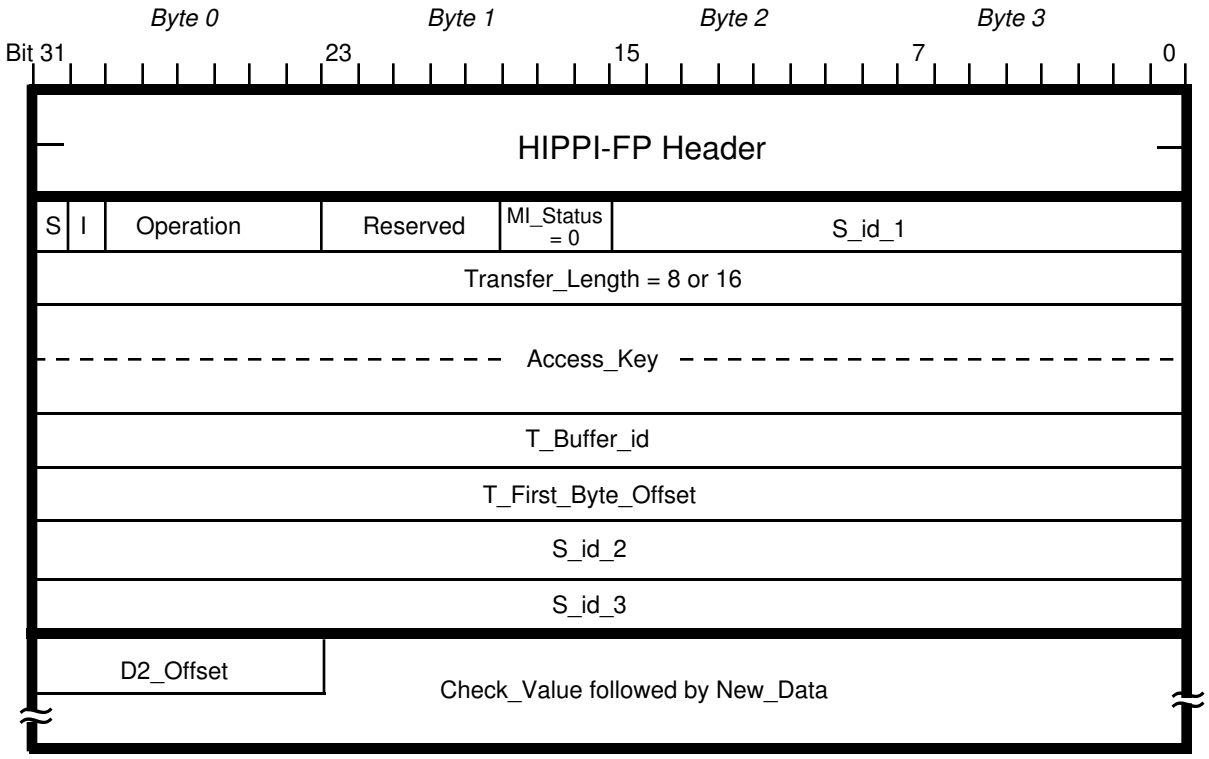

Figure 12 - HIPPI-MI lock command packet format

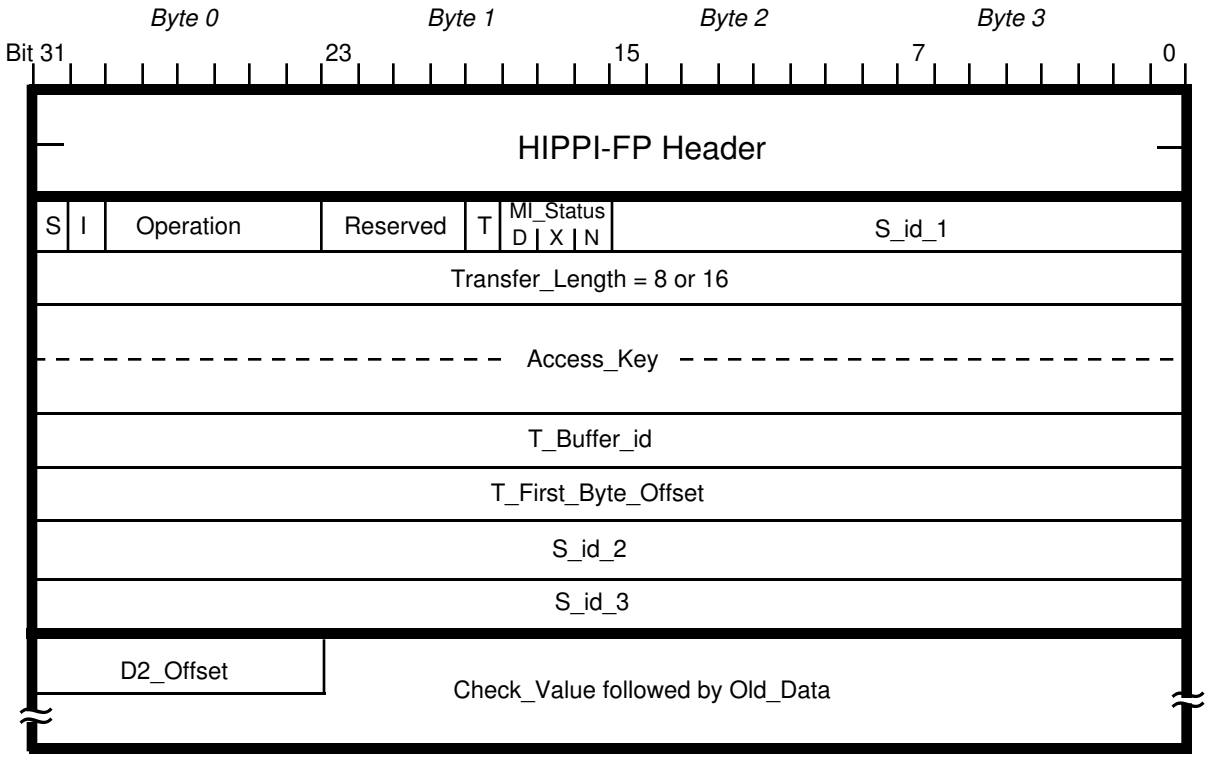

Figure 13 - HIPPI-MI lock reply packet format

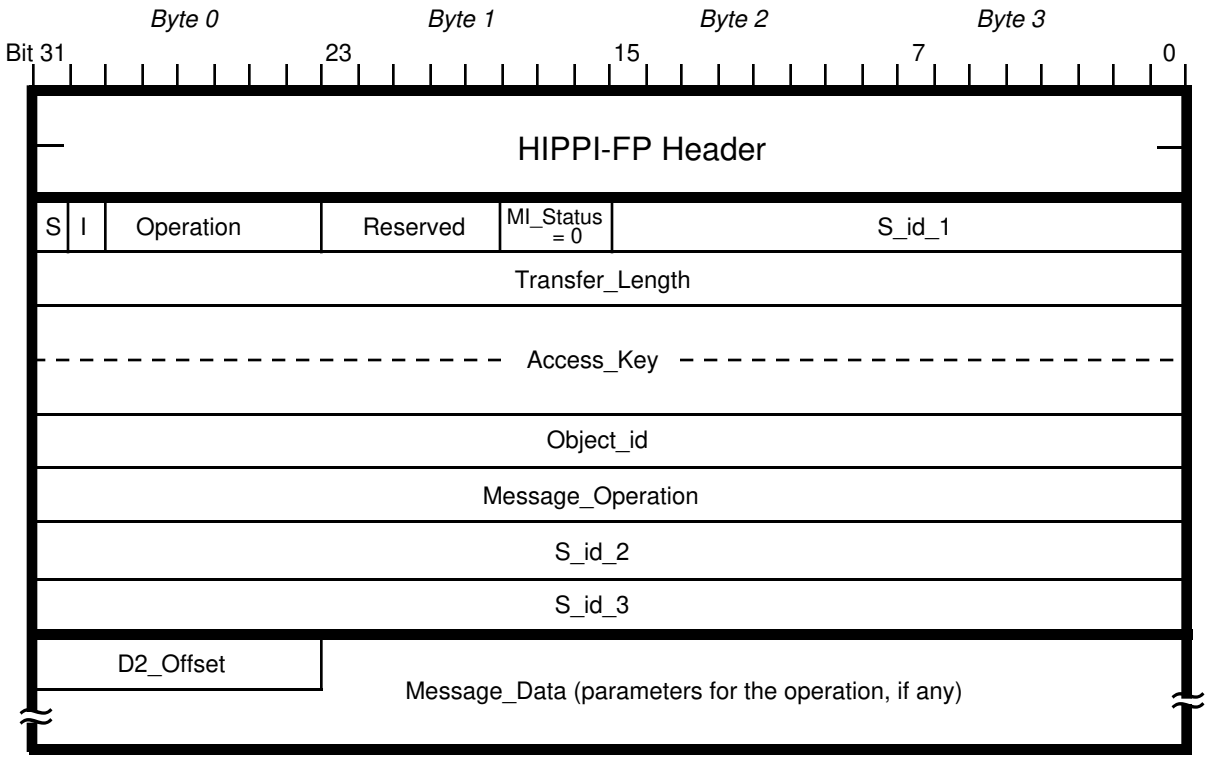

Figure 14 - HIPPI-MI message command packet format

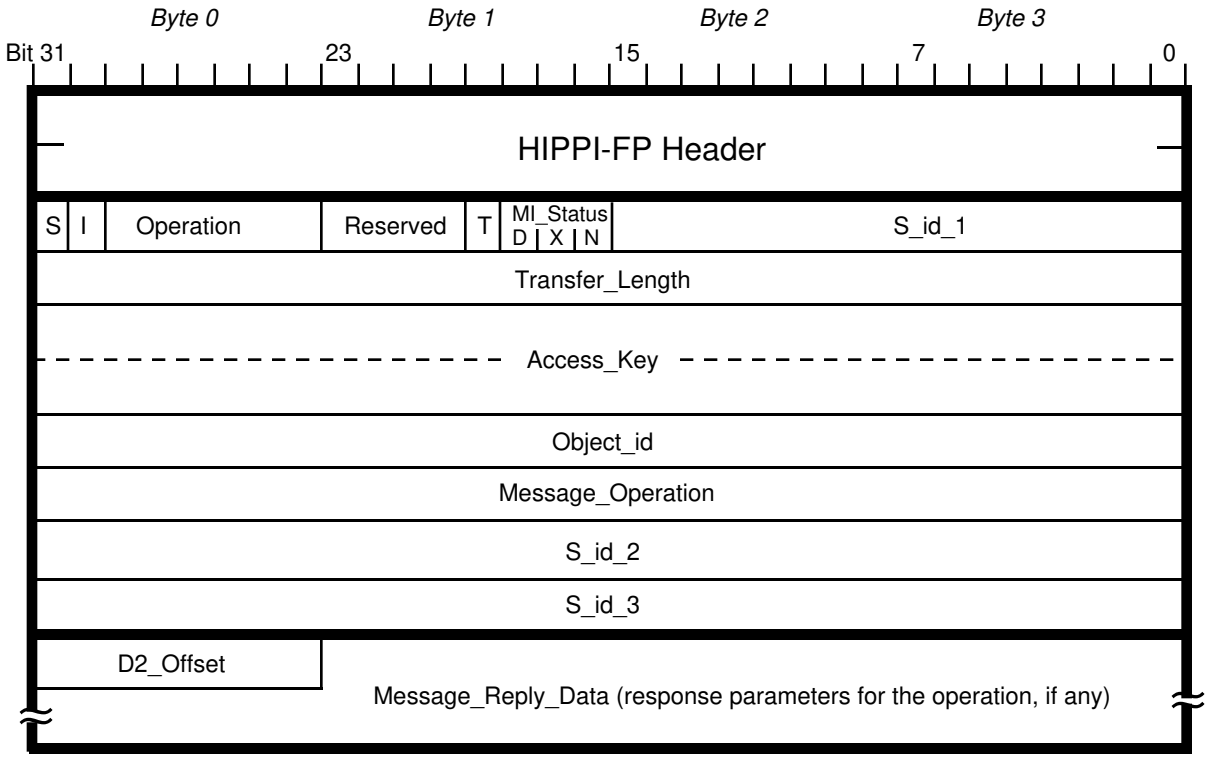

Figure 15 - HIPPI-MI message reply packet format