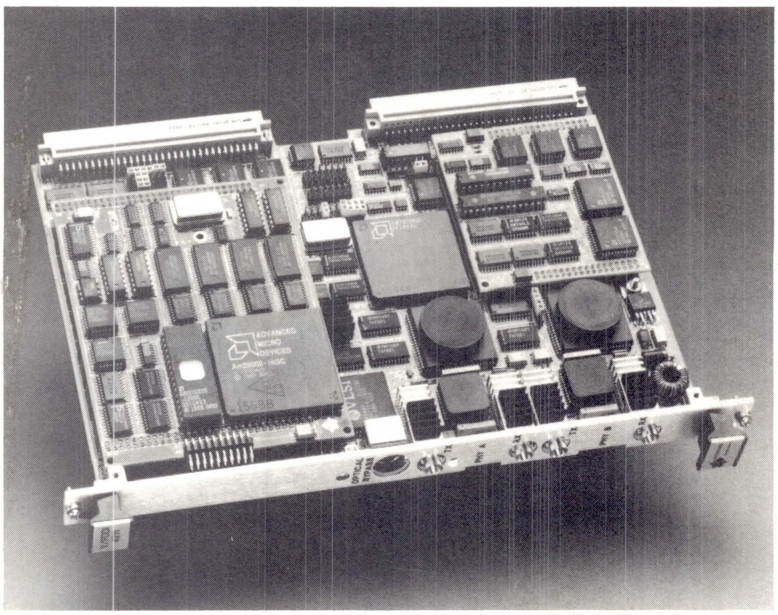


V/FDDI 4211 Peregrine

User's Guide

INTERPHASE
CORPORATION

V/FDDI 4211 Peregrine

High-Performance
FDDI Node Processor
for VMEbus

User's Guide

Document No.: UG04211-000,REVG
Release Date: July 20, 1992

COPYRIGHT NOTICE

INTERPHASE CORPORATION
13800 Senlac
Dallas, Texas 75234
Phone: (214) 919-9000
FAX: (214) 919-9200

DISCLAIMER

Information in this user document supersedes any preliminary specification, data sheets, and/or any other documents that may have been made available. Every effort has been made to supply accurate and complete information. However, Interphase Corporation assumes no responsibility or liability for its use. In addition, Interphase Corporation reserves the right to make product improvement without prior notice. Such improvements may include, but are not limited to, command codes and error codes.

FOR ASSISTANCE

To place an order for an Interphase product,
or for technical assistance, call:

CUSTOMER SERVICE:

In the U.S.: (214) 919-9111
In the United Kingdom: (869) 321222

TRADEMARK ACKNOWLEDGMENTS

All terms used in this manual that are known to be trademarks or service marks are listed below. In addition, terms suspected of being trademarks have been appropriately capitalized. Use of a term in this manual should not be regarded as affecting the validity of any trademark or service mark.

- SUPERNET™ is a trademark of Advanced Micro Devices.
- Interphase® is a registered trademark of Interphase Corporation.
- BUSpacket Interface℠ is a service mark of Interphase Corporation.
- ST™ is a trademark of AT&T.
- UNIX® is a registered trademark of AT&T Bell Laboratories.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

This page is intentionally left blank.

CHAPTER 1
INTRODUCTION

SCOPE

This manual is primarily intended for users who need to write a software driver for the V/FDDI 4211 Peregrine. It can also be used by individuals who need to install and/or test the 4211.

Readers are assumed to have extensive knowledge of the following:

- FDDI specifications, including revision 6.2 of FDDI Station Management specifications (ANSI X3T9.5/84-49)
- the C programming language, including experience writing and installing drivers
- the operating system of the host computer
- VMEbus specifications and hardware installation procedures

This manual's organization allows you to focus on your specific areas of interest, without giving you more information than needed. Specifically, it contains guidelines on:

- Installing the 4211
- Getting the board up and running
- Issuing commands to and obtaining responses from the board

HOW TO USE THIS MANUAL

You will find it very useful to read this *Introduction* completely. It contains information that will clarify many of your questions later. The *Conventions* section can be especially useful for later reference since it defines how certain topics will be presented to you.

Be sure to perform the installation of the product using the *Installation* chapter. Read this chapter thoroughly **before** attempting the installation.

Chapter 3 provides an overview of how the product's interface works. It gives procedures for submitting commands, as well as specific facts about the VMEbus and FDDI that affect board operation.

Chapter 4 describes the 4211 command set, ***excluding*** the generic RC Interface commands. The latter are documented in a separate manual from Interphase, the *Common Boot and RC Interface User's Guide* (see the *Related Publications and Standards* section, below). The Common Boot/RC manual describes the generic protocol used to boot and issue commands to a variety of Interphase controllers. Any parts of the Common Boot and RC Interface which do not apply to the 4211 will be noted where appropriate in this manual.

The *Specifications* chapter includes the 4211 specifications and connector pinouts. *Diagnostics* describes the 4211's power-up and post power-up diagnostics.

Two FDDI tutorials are provided in *Appendix A* for those readers who are not intimately familiar with FDDI. This section can even be useful for experienced users because it standardizes terminology.

Appendix B contains C data structures from the 4211's on-board firmware. *Appendix C* provides a table showing the jumper settings for all possible short I/O base addresses on the 4211. *Appendix D* describes how to download code to 4211 firmware for execution.

CHANGES FROM PREVIOUS REVISION

This section applies to users who are familiar with the previous revision of this manual (UG04211-000,REVF). The following list summarizes how the current revision of the manual differs from the previous one. Each item in the list includes a reference to page(s) affected by the change.

1. In Diagram 4-4 on p. 4-60 (Returned Data in CAM Command Register), change the note to state that the bits which are not referenced in the diagram are always 0x0. Previously, the value of these bits was not guaranteed.

2. In Diagram 4-5 on p. 4-61 (Returned Data in Front End Command Register), change the note to state that the bits which are not referenced in the diagram are always 0x0. Previously, the value of these bits was not guaranteed.

3. In Table 4-21 on p. 4-62 (Returned Optical Bypass Control Modes), update the table of returned codes as follows:

CODE	MODE
0x0	Optical Bypass Relay not present
0x1	Optical Bypass Relay present and bypass mode enabled
0x2	Optical Bypass Relay present and THRU mode enabled

RELATED PUBLICATIONS AND STANDARDS

The following publications are excellent resources of information when working with this product. If you do not have access to any of the Interphase publications, call us.

- *Common Boot and RC Interface User's Guide*
  Interphase Corporation
  Dallas, TX 1991
  Documents a generic protocol for booting and issuing commands to a family of Interphase controllers, including the V/FDDI 4211 Peregrine.

- *The SUPERNET™ Family for FDDI Databook*
  Advanced Micro Devices
  Sunnyvale, CA 1989
  Specifications for individual members of AMD's SUPERNET family chip set.

- *SUPERNET™ Technical Manual*
  Advanced Micro Devices
  Sunnyvale, CA 1989
  A detailed description of the SUPERNET chip set and its relationship to FDDI.

- *The VMEbus Specification*
  Micrology pbt.
  Printex Publishing Inc.
  Revision C.1 (October 1985)
  VMEbus specifications and terminology.

- *Writing a Unix® Device Driver*
  Janet I. Egan/Thomas J Teixeira
  John Wiley and Sons, Inc.
  1988
  An excellent introduction to writing Unix drivers. It shows how Unix performs input/output, how device drivers relate to the hardware I/O architecture, and how to incorporate the device driver into the Unix kernel. It also contains an excellent discussion of how to test drivers.

- FDDI references:
  - ANSI X3T9.5/84-49 FDDI Station Management (SMT) - Rev 6.2, May 18, 1990
  - ANSI X3T9.5 FDDI Token Ring Media Access Control (MAC) - Rev 10, Feb. 28, 1986
  - ANSI X3T9.5 FDDI Physical Layer Protocol (PHY) - Rev 15, Sept. 1, 1987
  - ANSI X3T9.5 FDDI Physical Layer Medium Dependent (PMD) - Rev 7, Feb. 20, 1987

  The FDDI specifications listed above may be ordered from either of the following organizations:

  Primary source:
  Global Engineering
  2805 McGraw Ave.
  Irvine, CA 92714
  (800) 854-7179

  Secondary source:
  American National Standards Institute
  1430 Broadway
  New York, NY 10018
  (212) 354-3300

PRODUCT OVERVIEW

The V/FDDI 4211 Peregrine is a RISC-based high-performance node processor for 100 Megabit-per-second fiber optic FDDI (Fiber Distributed Data Interface) networks. Besides connecting a workstation or computer system based on VMEbus to an FDDI network, the 4211 Peregrine can perform much of the communications protocol processing, as well as certain network management functions required by FDDI.

Design Flexibility

The flexibility of the 4211 Peregrine's design allows it to be configured to support any FDDI implementation, including the use of FDDI as a high-speed local area network or as a backbone connecting other networks.

6U or 9U Compatibility

Because of its 6U form factor, the 4211 can be integrated into 6U and, with the use of an adapter, 9U VMEbus card cages.

Single Attachment or Dual Attachment

The 4211 supports both Class A Dual Attachment and Class B Single Attachment stations as defined in ANSI's FDDI specification. As more FDDI networks are installed, an increasing number of them are expected to use hybrid configurations of both single- and dual-attachment stations combined with bridges and routers to other types of networks such as Ethernet or Token Ring. An example of such a hybrid system is shown in Figure 1-1.

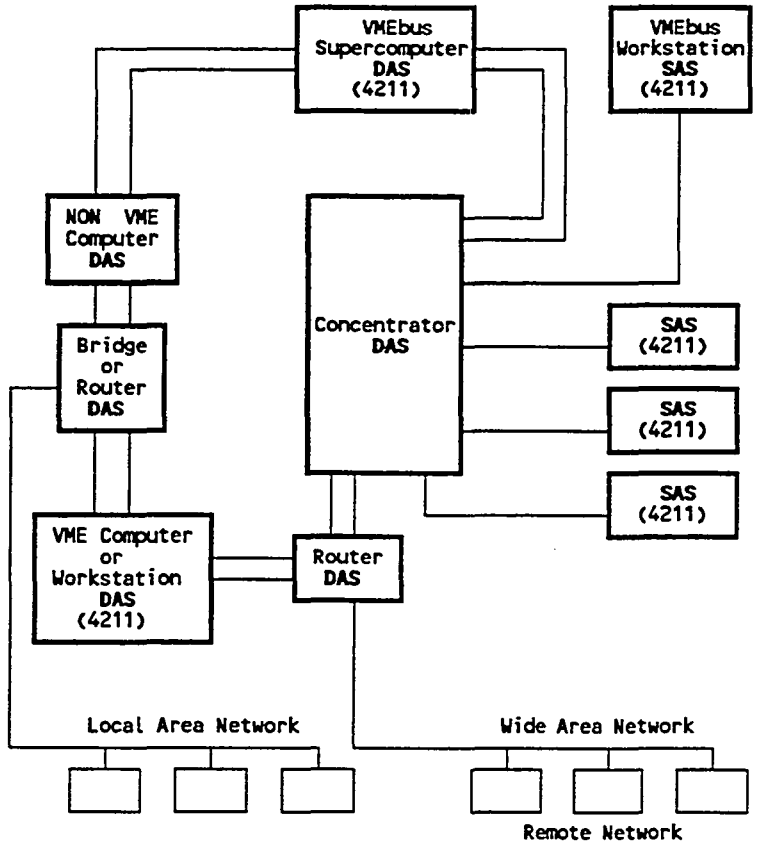


Figure 1-1 . Hybrid FDDI Network

Single- or Dual-MAC

The 4211 supports either single or dual MACs (Media Access Control), in compliance with the FDDI specification. For additional information on single vs. dual MAC implementations, refer to the tutorials in Appendix A.

Overview of Hardware/Firmware Design

Key design features of the V/FDDI 4211 include:

- AMD SUPERNET™ Chip Set
- Am29000 RISC processor
- 1-Megabyte Communications Buffer
- BUSpacket Interface℠
- SMT Functions in On-Board Firmware

Figure 1-2 (p. 1-7) is a block diagram of the V/FDDI 4211. Later sections contain descriptions of the various points of interest.

AMD SUPERNET™ Chip Set

For its interface to the FDDI network, the 4211 utilizes members of AMD's SUPERNET chip set. The first FDDI chip set to come to market, the SUPERNET family is widely used in currently available FDDI products.

On-Board RISC Processor

The 4211 uses AMD's Am29000 RISC processor, which is capable of operating at 16 MIPS (millions of instructions/second) in this type of application. To maximize performance, the 4211 includes 512 Kbytes of dedicated execution RAM. This frees the 29000 from having to contend with the FDDI or VMEbus interfaces for accesses to memory, and thereby enables the 29000 to execute at rates approaching its peak performance.

If desired, the host can download protocol software to the 4211 for execution by the 29000. The 4211's firmware is based on the standard Unix® streams environment, which simplifies the process of porting protocol stacks to the 4211.

1M Communications Buffer

The 4211 has a one-megabyte communications buffer. A buffer of this size is particularly useful in a high-performance networking environment, since it virtually eliminates the data overruns associated with smaller buffers. Most competing FDDI node processors must perform a second operation to access communications buffers beyond 256 Kbytes.

256-Entry CAM

The 4211 supports on-board content-addressable memory (CAM). This feature enables the board to receive frames addressed to as many as 256 different destinations, in addition to the normal base station address embedded in NOVRAM.

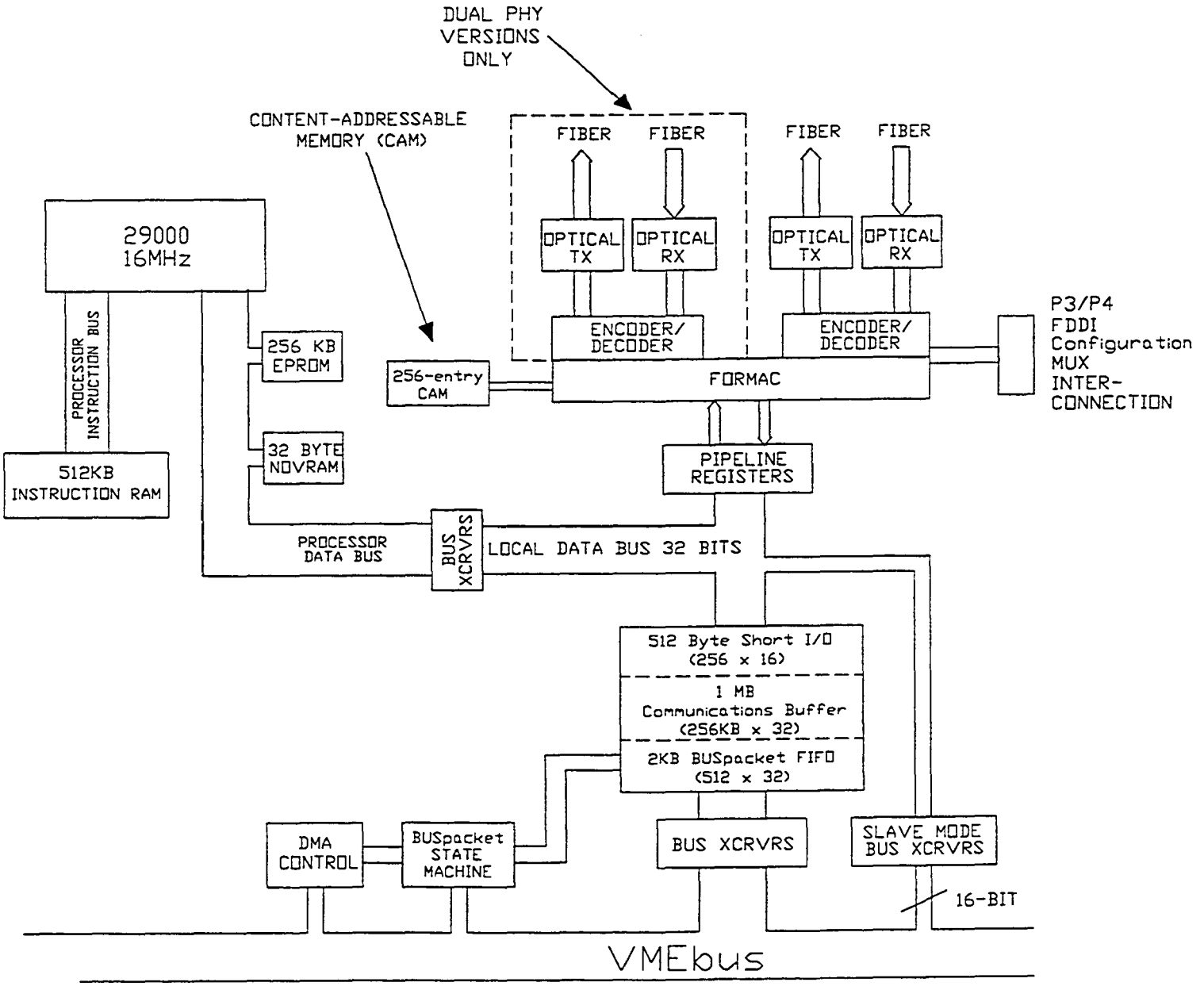


Figure 1-2 . V/FDDI 4211 Block Diagram

High-Performance VMEbus Interface

The 4211 features Interphase's exclusive BUSpacket Interface[SM] for its interface to the VMEbus. Capable of performing VMEbus data transfers of over 30 MBytes/sec., the 4211 has the fastest DMA interface available on the VMEbus.

Essentially, the BUSpacket Interface decouples VMEbus activity from other activity on the 4211 through the use of video RAMs and an asynchronous state machine that monitors VMEbus signalling. The BUSpacket Interface formats packets of data in the FIFO side of the video RAM before seizing control of the VMEbus. Then, the FIFOs are emptied at speeds that are only limited by the speed of the host system's memory.

The FIFO Buffer. The BUSpacket Interface has a very high speed 2K FIFO. It can write a BUSpacket to (or read a BUSpacket from) the V/FDDI 4211's communications buffer with a fast VMEbus Block Transfer. Because the FIFO itself is very fast, once it is connected to the VMEbus, the DMA can proceed at speeds exceeding 30 megabytes/second. Thus, the speed of the 4211 on the VMEbus is totally independent of the speed of the communications buffer.

The BUSpacket State Machine. Synchronous state machines are typically used to control the entire DMA operation. They must first synchronize the inherently asynchronous VMEbus control signals to avoid metastable states. This synchronization cycle, on the average, delays the signal by one and a half periods of the synchronizing clock. For example, a synchronous state machine running at 20 Mhz delays any control signal off the bus by an average of 75 nanoseconds before it uses the signal as an input. Such a delay is an unacceptable portion of the 100 nanosecond cycle time required to achieve a 40 megabyte/second data rate on the VMEbus.

In contrast, the high-speed DMA transfer of the BUSpacket Interface is controlled by an asynchronous state machine that runs off a tapped delay line. This configuration avoids metastable states without incurring the delays caused by synchronizing the VMEbus signals. Since the VMEbus is an asynchronous bus, the tapped delay line state machine can essentially drive the bus as fast as system memory allows.

The VMEbus interface of an FDDI network controller must be able to transfer data at rates significantly faster than the FDDI network. Otherwise, data coming from the network to the host will "hog" bus bandwidth. The 4211's VMEbus data transfer capabilities approach the 40 Megabyte/sec. theoretical limit of the VMEbus specifications. This high-speed operation is critical in FDDI applications, due to the speed of the FDDI network (100 Megabits/sec. or 12.5 Megabytes/sec.).

On-Board Station Management (SMT)

The 4211 supports FDDI Station Management capabilities in on-board firmware. By running SMT protocols on-board, the 4211 frees the host to perform other tasks.

FDDI's SMT can be thought of as a vertical communications protocol environment stretching across the first four levels of the OSI models. It includes functions such as connection management, which affects ring formation and fault isolation/recovery, and frame-based protocols for network management. For more information, see the FDDI tutorial in Appendix A.

CONVENTIONS

This section details many of the writing conventions used throughout the manual. In addition, it gives many of the technical conventions.

- The terms "controller" and "4211" are synonymous.
- The term "short I/O" refers to a VMEbus-addressed block of memory in which only the lower 16 VMEbus address lines are used for transactions. The upper 16 VMEbus address lines are not used.
- "Byte" represents 8 bits; "word" represents 16 bits (2 bytes); and "long-word" represents 32 bits (2 words, 4 bytes).

  NOTE: In AMD 29000 documentation, a "word" is defined to be 32 bits long. This definition is **not** used in this manual.

- Binary (single bit) data is represented as either '1' or '0'.
- When used in the context of a single bit of data, the term "set" means that the bit is a one ('1').
- Similarly, the term "cleared" means that the bit is a zero ('0').
- To represent hexadecimal numbers, the manual adopts the C language notation. Decimal numbers are shown as decimal digits. For example:

  0x29 = 29 hex
  41 = 41 decimal

- Many commands contain bits, bytes, or words which are marked "RESERVED". Such reserved fields fall into four categories:

  [1] Reserved for future use, must be cleared ('0', 0x00, or 0x0000).

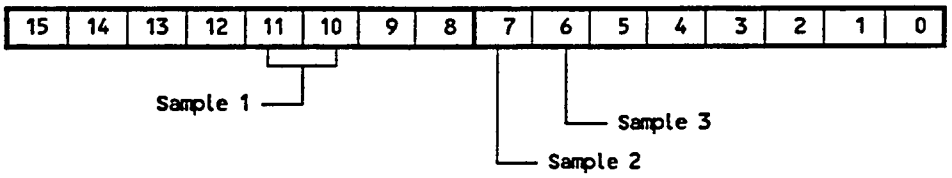
  [2] Reserved for future use, do not write to this field.

  [3] This field is reserved for a future implementation of the "pools" concept. It must be cleared ('0', 0x00, or 0x0000) under the current implementation.

  [4] This field is reserved for a future implementation of controllers with dual-MAC capability. It must be cleared ('0', 0x00, or 0x0000) under the current implementation.

  In this manual, whenever a data structure contains reserved field(s) it will also contain a statement about how the host should treat the reserved field(s). The statement will be placed immediately after the data structure.

- When showing binary representations of bytes or words, the diagrams may have many bits which do not have names. These are RESERVED. As an example:



Bits 10 and 11 are called Sample 1, bit 7 is called Sample 2, and bit 6 is called Sample 3. All other bits are RESERVED.

CHAPTER 2
INSTALLATION

OVERVIEW

Before attempting installation, read this chapter thoroughly to insure the safe installation of the 4211 into your system. If you have any questions regarding installation which are not answered in this chapter, please contact Interphase Customer Service at (214) 919-9111.

The 4211 is installed into the VMEbus system using the following steps:

1. Visual Inspection
2. Set On-Board Jumpers
3. Power System Off
4. Install Board
5. Cabling Procedure
6. Power System On

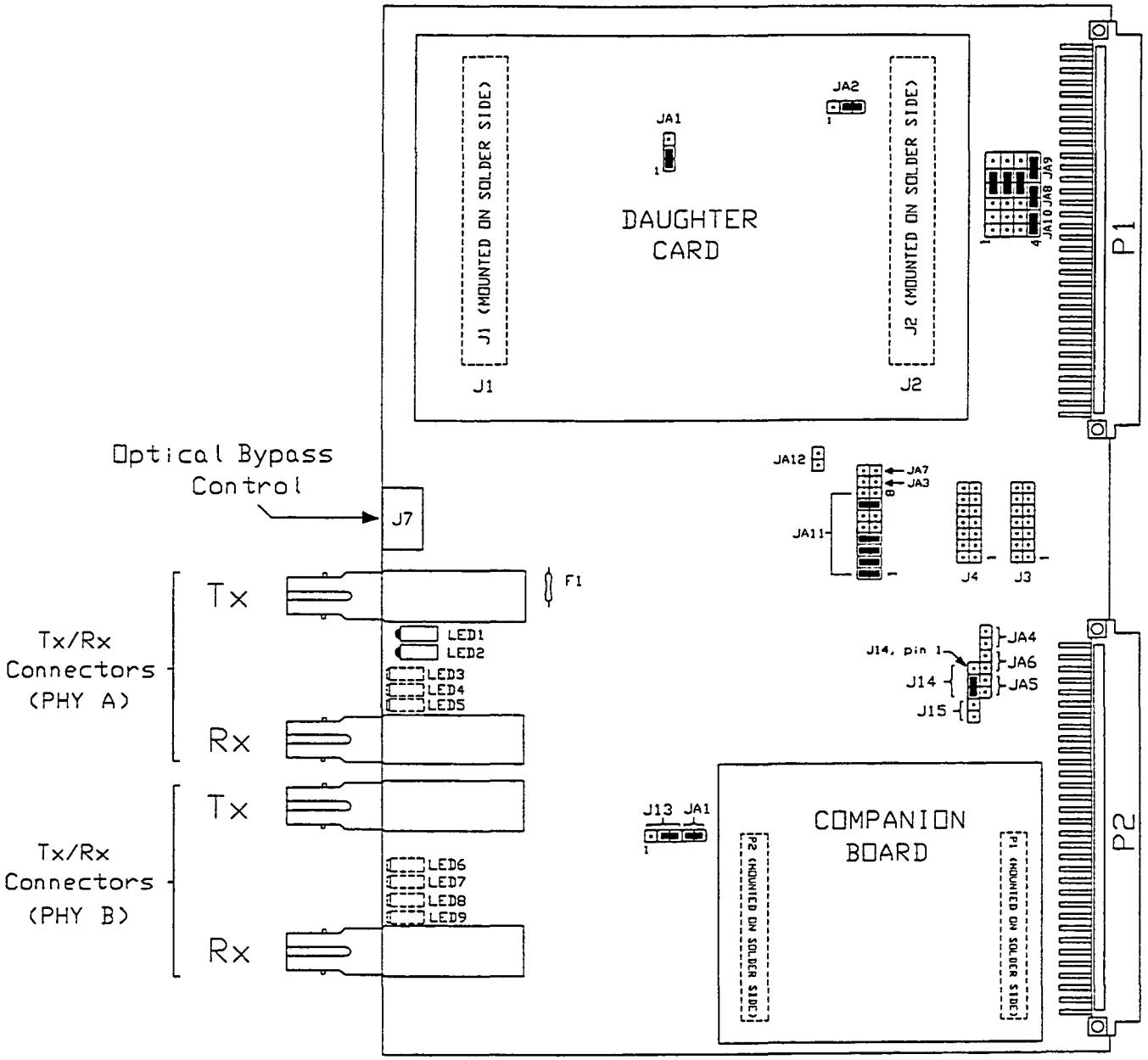
When installing the 4211, the following **WARNING** must be adhered to.

WARNING

1. *Do NOT install or apply power to a damaged board. Failure to observe this warning could result in extensive damage to the board and/or system.*

2. *Caution! The 4211 is sensitive to electrostatic discharge (ESD), and the board could be damaged if handled improperly. Interphase ships the board enclosed in a special anti-static bag. Upon receipt of the board, take the proper measures to eliminate board damage due to ESD (i.e., wear a wrist ground strap or other grounding device).*

The figure on the next page shows the 4211 board layout. Please refer to it as necessary when following the step-by-step installation procedure.



JUMPER SETTINGS

= JUMPER INSTALLED.

= NO JUMPER INSTALLED.

NOTES:

[1] Versions of the 4211 designed for use as a single-attachment station only have one pair of optical connectors (labelled 'PHY A' above). For additional information on the PHY A / PHY B designation given above, please see 'Considerations for Cabling a Dual-Attachment Station' in this chapter.

[2] The jumper settings on your board may be different from those shown above.

[3] A second fuse, F2, is located beneath the companion board.

[4] LED3 - LED9 are provided on boards with hardware revision level HD-4211-*xxx*,REVD and earlier. Unlike LED1 and LED2, these LEDs are surface-mounted to the back of the board and are not readily visible unless they are lit up.

Figure 2-1 . V/FDDI 4211 Peregrine Board Layout

INSTALLATION PROCEDURE

The following is a step-by-step procedure for installing the 4211.

Step 1. Visual Inspection

Before attempting the installation of this board, make sure you are wearing an anti-static or grounding device. Remove the 4211 board from the anti-static bag, and visually inspect it to ensure no damage has occurred during shipment. A visual inspection usually is sufficient, since each board is thoroughly checked at Interphase just prior to shipment.

If the board is undamaged and all parts are accounted for, proceed with the installation.

Step 2. Set On-Board Jumpers

Set all on-board jumpers so that the 4211 is properly configured for operation within your system. The jumpers are summarized in the following table. To locate the jumpers, refer to the board layout on page 2-2.

Table 2-1 . Summary of 4211 Jumpers

MOTHER BOARD JUMPERS	
Jumper	Function
JA1	Single/Dual PHY
JA2	(not used)
JA3	BCLK Termination
JA4	Local Clock
JA5	Optical Bypass Control
JA6	" " "
JA7	Early BBSY* Release
JA8	Bus Request Level
JA9	" " "
JA10	" " "
JA11 (pins 1-7)	Short I/O Base Address
JA11 (pin 8)	VME Address Modifiers
JA12	Factory Test *
J13	Missed Frame Interrupt
J14	Frame Segmentation
J15	Restricted Token Interrupt
DAUGHTER CARD JUMPERS	
Jumper	Function
JA1	EPROM Size
JA2	Single vs. Split Export Register

* Jumper must be REMOVED for all board versions.

As discussed in the remainder on this section, some of the jumpers have default settings which should not (or cannot) be altered. The remaining jumpers can be used to help configure the board for your specific system.

Motherboard Jumper Settings

Single/Dual PHY. A jumper is installed in JA1 if your version of the 4211 includes the hardware necessary to implement a second PHY. If your board only supports a single PHY, there should not be a jumper in JA1. This jumper is set at the factory, and it should not be changed.

Local Clock. Jumpers JA3 and JA4 concern the local clock used in dual-MAC configurations. **Single-MAC users should NOT have either of these jumpers installed** - they are for dual-MAC applications only.

Installing a jumper in JA3 enables byte clock (BCLK) termination. Installing a jumper in JA4 disables the local clock. In a dual-MAC configuration, one of the 4211s must have JA3 and JA4 INSTALLED. The other 4211 must have these jumpers REMOVED. The net effect is that one board supplies the clock, and the other terminates it. It makes no difference which board has the jumpers installed.

Optical Bypass Control. Jumpers JA5 and JA6 disable host control of the optical bypass switch. If the jumper is installed in JA5, the host will *not* be able to control the bypass switch for the primary PHY. Likewise, if the jumper is installed in JA6, the host will not be able to control the bypass mechanism for the secondary PHY. Installing either jumper disables host control of the bypass switch. In other words, the host will be unable to remove the station (4211) from the FDDI ring. The default setting for JA5 and JA6 is jumpers removed.

Early vs. Late BBSY* Release. Jumper JA7 determines whether the 4211 will use early or late release VMEbus arbitration when it is the bus master. If the jumper is set for late release (jumper IN), the VMEbus signal BBSY* is released **after** the last cycle is completely finished. If the jumper is set for early release (jumper OUT), BBSY* is released **at the start** of the last cycle to allow for rearbitration during the last cycle.

VMEbus Request Level. Jumper fields JA8, JA9, and JA10 are used to set the 4211's VMEbus request level from 0 (lowest) to 3 (highest).

Figure 2-2 shows all possible valid configurations for this jumper block.

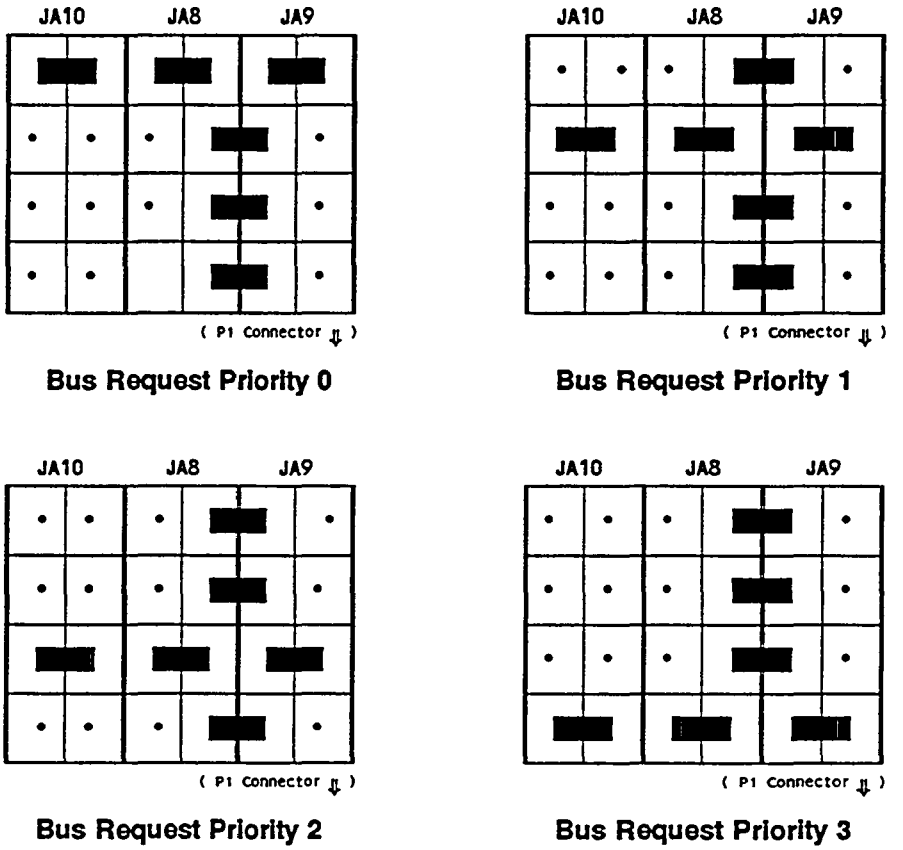


Figure 2-2 . VMEbus Request Priority Jumper Settings

Short I/O Base Address. Pins 1 through 7 of jumper field JA11 are used to set the base address of the 512 bytes of short I/O space RAM on the 4211. As discussed in the Functional Overview, all interaction between the host and the 4211 takes place in this 512-byte space.

The jumpers correspond to VMEbus address lines A9-A15, respectively, as shown in the following table:

Table 2-2 . Jumpers Used for Short I/O Base Address

JA11 Pin #	VMEbus Address Bit
7	A15
6	A14
5	A13
4	A12
3	A11
2	A10
1	A9

Removing a jumper sets the corresponding bit to '1'. Installing a jumper clears the address bit to '0'. The short I/O base address must be a multiple of 0x200.

For example:

Jumper Setting*	VMEbus Address Bits**							
7654321	15	14	13	12	11	10	9	Base Address
IIIIOII	0	0	0	0	1	0	0	0x0800
IOIOOIO	0	1	0	1	1	0	1	0x5A00
IOOIIII	0	1	1	0	0	0	0	0x6000
OIOIOIO	1	0	1	0	1	0	1	0xAA00

* I = Jumper IN
O = Jumper OUT

** All bit definitions are binary ('0' or '1').
Address bits 8 through 0 are '0'.

Address Modifiers Allowed In Short I/O. Jumper pin 8 in JA11 is used to select the address modifiers that are permitted in the short I/O (A16) address space. If the jumper is installed, only short supervisor accesses are permitted (address modifier 0x2D only). If no jumper is installed, then both supervisor and user accesses (0x2D and 0x29) are allowed.

Test Jumper. Jumper JA12 is used for factory test purposes. It should be left in its default setting (jumper OUT).

Missed Frame Interrupts. Jumper J13 concerns your board's ability to generate an interrupt in the event of a missed frame. This is determined by on-board firmware, so the jumper is not user-settable. If jumper pins 1 - 2 are jumpered together, the interrupt is enabled. If pins 2 - 3 are jumpered together, the interrupt is disabled. Do not change the factory setting.

Frame Segmentation. Jumper J14 concerns your board's ability to break up frames into 2K segments. If this feature is supported, it allows software to allocate smaller buffers and use only as many buffers as are required for the size of the frame. The first boundary in a frame will be at 2K minus 256 bytes; subsequent boundaries will be at 2K multiples. This feature is controlled by on-board firmware, so the jumper is not user-settable. If jumper pins 1 ↔ 2 are jumpered together, frame segmentation is enabled. If pins 2 ↔ 3 are jumpered together, segmentation is disabled. Do not change the factory setting.

Restricted Token Interrupts. Placing a jumper in J15 enables restricted token interrupts (as documented in SMT 6.2 section 10.4.1.4). Removing the jumper disables such interrupts. Restricted tokens are not currently supported by the 4211's firmware. Therefore, J15 must be left in its default setting (jumper OUT).

Daughter Card Jumper Settings

EPROM Size. If your version of the 4211 has a 32-pin EPROM installed, pins 1 and 2 of JA1 will be jumpered together at the factory. If your board has a 28-pin EPROM, the factory will instead place a jumper on pins 2 and 3. Do not change the factory-provided jumper setting.

Single vs. Split Export Register. The setting of jumper JA2 depends on whether your controller has a single or split export register. If it has a single export register, pins 2 and 3 will be jumpered together at the factory. Otherwise, pins 1 and 2 will be jumpered together. Do not change the factory-provided jumper setting.

Step 3. Power System Off

Ensure that the host system and peripherals are turned OFF.

CAUTION

System power and peripheral power must be turned OFF before attempting to install the 4211. Failure to do so may result in severe damage to the board and/or system.

Step 4. Install Board(s)

Ensure that system power is OFF.

If you are setting up a single-MAC station (single-attachment or dual-attachment), carefully slide the 4211 into the VMEbus card slot. It should slide all the way in without any difficulty. If it doesn't, pull it out and check to make sure that there are no cables in the way.
If you are setting up a dual-MAC, dual-attachment station, follow Step 1 in the cabling procedure described on p. 2-21. Then carefully slide the two interconnected 4211s into *adjacent* VMEbus slots.

If you are setting up a dual-MAC, dual-attachment station, follow Step 1 in the cabling procedure described on p. 2-21. Then carefully slide the two interconnected 4211s into *adjacent* VMEbus slots.

Once the board(s) are properly seated in the slot(s), tighten the captive mounting screws on each end of the board front panel(s).

Step 5. Cabling Procedure

This section provides generic cabling instructions. Its purpose is to give you a general understanding of how to set up various types of FDDI stations using the 4211. Be aware, however, that *many other configurations are possible*. For information on selecting the optimum cables and connectors for your installation, contact your FDDI cable manufacturer.

The cabling procedure varies depending on whether you are setting up a:

- single-attachment station (see p. 2-15)
- single-MAC, dual-attachment station (see p. 2-17)
- dual-MAC, dual-attachment station (see p. 2-21)

Required Cables and Connectors

To save board space, the 4211 has ST™-compatible optical connectors instead of FDDI-standard MIC (Media Interface Connector) receptacles. Therefore, all FDDI cables (and related components such as the optical bypass switch, if needed) require bayonet connectors on the ST (board) end in order to mate with the 4211's Tx/Rx connectors. Figure 2-3 depicts both connector types.

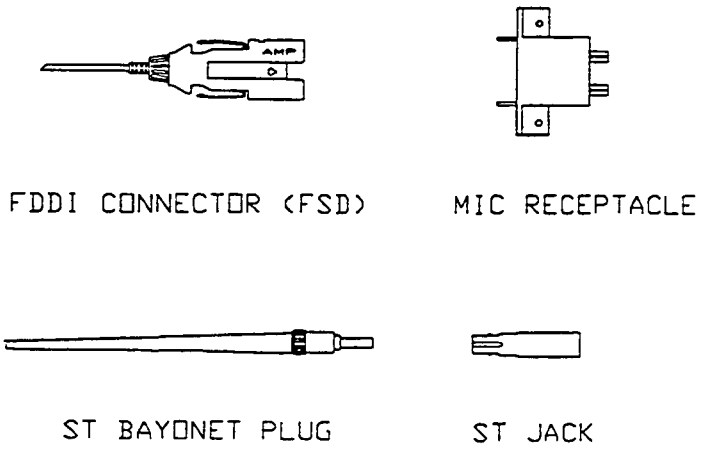


Figure 2–3 . ST- and FDDI-Standard Connectors

The following page lists sample part numbers of components which can be used to set up various types of FDDI stations. These components will be referred to in the cabling instructions given in the remainder of this chapter.

Table 2-3 . Sample Parts for Cabling the 4211

Type of Part [1]	Sample Part Number	Purpose
ST-to-MIC Cable	AMP 502125-7 [2]	Routes Tx and Rx signals from the ST connectors of one 4211 PHY. The FSD connector on the cable plugs into the MIC receptacle of a concentrator.
MIC-to-MIC Cable	AMP 502122-7 [3]	In the subsections describing how to cable the different types of dual-attachment stations (see pp. 2-17, 2-21), this cable connects the optical bypass switch to the data link.
Optical Bypass Switch	Interphase Part # [4] SA0000070	Provides the optical bypassing mechanism described in FDDI specifications.
2 Dual MAC Cables	N/A[5]	Interconnects the J3 and J4 connectors of two 4211s to implement a dual-MAC, dual-attachment station.

NOTES:

[1] These components are *representative* of the types of parts used to cable various types of FDDI stations. Contact your FDDI cable manufacturer for details on selecting the optimum cables and connectors for your installation.

All FDDI connectors must be correctly keyed for your specific setup (i.e. M-S key for a single-attachment station, or A-B for a dual-attachment station). In general, you can specify the desired keying when ordering your cables.

[2] This part number is for a 15-meter dual-fiber cable with two 2.5mm bayonet ST connectors on one end and an FDDI connector on the other. Its cable size is 62.5/125 μm.

[3] This part number is for a 15-meter light-duty dual-fiber cable with two FDDI connectors at each end. Its cable size is 62.5/125 μm.

[4] This part is an AMP 502443-3 optical bypass switch. The switch has two pairs of ST connectors that mate with the Tx/Rx connectors on a dual-PHY 4211. It also has a 6-pin mini DIN plug on its optical bypass control cable that mates with the 4211's J7 connector. If you are setting up a dual PHY, dual MAC station, the optical bypass control cable must be modified to route control signals to/from both PHYs.

[5] These cables are required in dual PHY, dual MAC (DASDM) configurations only. No sample part number is available. Make two 3"-long cables using 14-wire, flat-ribbon cable (3M 3365/14). On each end of the cables, attach a 14-pin IDC female connector (3M 3385-6614). All signals are one-to-one. For pinouts of the J3 and J4 connectors, see pp. 5-10 - 5-11.

Considerations for Cabling a Dual-Attachment Station

When cabling the 4211 as a dual-attachment station, it is essential to pay close attention to how Tx/Rx signals are routed to/from the board. For example, assume that you have three single-MAC, dual-attachment stations connected to form a standard, counter-rotating FDDI ring. Assuming that no fault has occurred, the flow of Tx and Rx signals between PHYs is as follows:

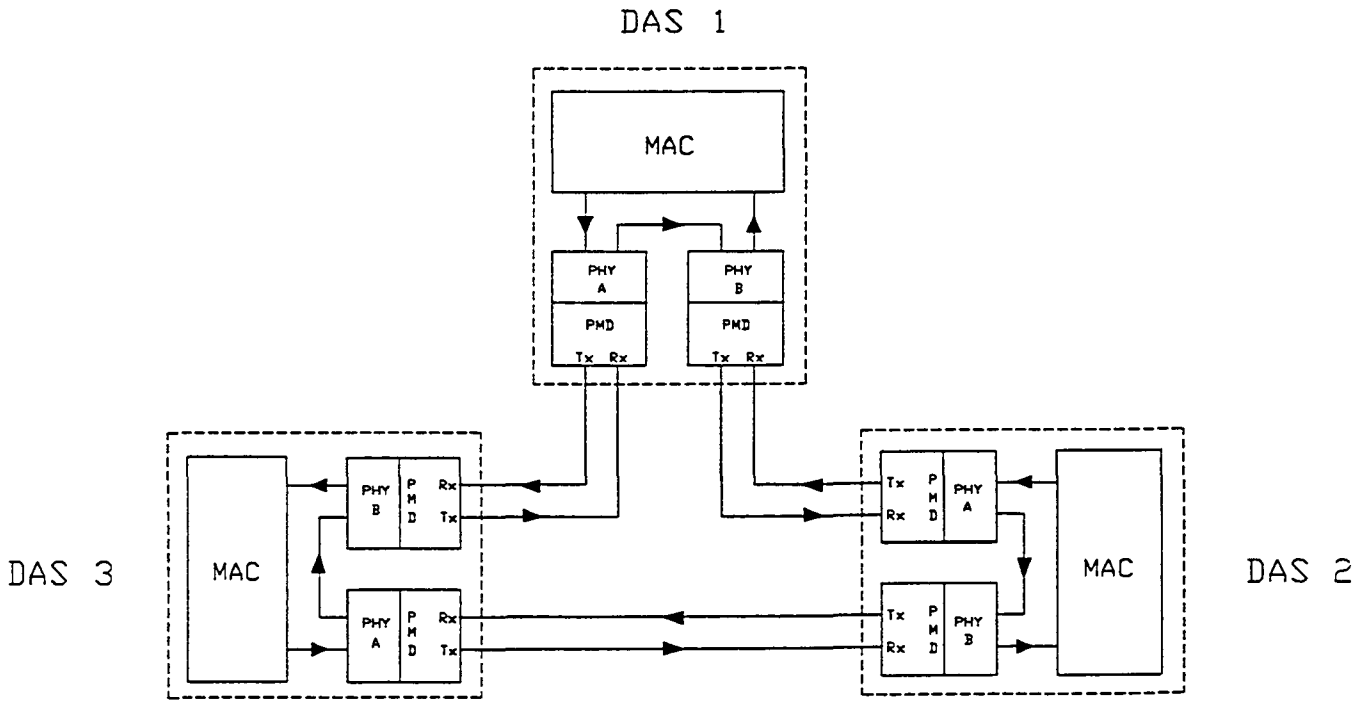


Figure 2-4 . Flow of Tx/Rx Signals between PHYs of Single-MAC, Dual-Attachment Stations (No Fault in Ring)

Notice in Figure 2-4 that outgoing data from one station's PHY A feeds into its downstream neighbor's PHY B as incoming data. Likewise, data that comes in through a station's PHY B goes out through its PHY A. As discussed at length in the FDDI specifications, this allows the ring to be reconfigured if a fault occurs.

PHY A vs. PHY B on the 4211

Versions of the 4211 designed for use as a dual-attachment station have two Tx/Rx connector pairs. From a hardware standpoint, either of these connector pairs could be used as PHY A or PHY B.

However, you should be aware that the board layout on p. 2-2 associates the two Tx/Rx connector pairs with a specific PHY. This is because certain components of the 4211's firmware

expect the host system to use this designation. It also matches the lettering printed on 4211 front panels available from Interphase.

In some networks, it may be necessary to reverse this designation and use the Tx/Rx connector pair labelled "PHY A" as PHY B, and vice versa. If you do so, make sure that all affected components of the FDDI system are modified accordingly.

Keying FDDI Connectors

Before cabling the 4211 as a dual-attachment station, it will be necessary for you to key some of the FDDI-standard connectors on the cables. For example, the ST-to-MIC cable specified in Table 2-3 (p. 2-9) is one of AMP's "universal" FDDI cables. As delivered to the user, the FDDI end of the cable will mate with *any* MIC receptacle.

> **IMPORTANT**
> It is essential for all FDDI connectors to be properly keyed. Otherwise, it is likely that signals will get crossed during installation or when network cables are later detached/reconnected.

Deciding How to Key the Connectors. For information on how the FDDI connectors of a dual-attachment station should be keyed, refer to the appropriate cabling diagram in this chapter (p. 2-16 for single MAC and p. 2-19 for dual MAC). These drawings are informative even if you are not using the sample parts given previously in this chapter.

The drawings only show keying which is local to the 4211 station. The connectors required by other physical entities in the network will vary depending on your setup.

Identifying and Installing Connector Keys. To determine the keying of an FDDI connector, examine it for some kind of identifying mark or aperture for the key. For example, suppose that you are looking at an AMP FDDI cable connector. Pull off the protective cover and locate the key aperture, which is approximately 1/4" from the end of the connector. If the aperture is empty, the connector is unkeyed ("universal"). If a key is installed, you can identify it by its color (see Table 2-5).

Table 2–5 . Keying Scheme on AMP FDDI Connectors

KEY COLOR	KEY TYPE	FUNCTION
Red	A	Connect to PHY A
Blue	B	Connect to PHY B
Green	M	Connect to concentrator or single-attachment station

To key an unkeyed AMP FDDI cable connector:

1. Get the desired key. Keys for an AMP FDDI connector are stored in connector's protective cover. To remove a key from the protective cover, pry it out carefully with a pen knife.

2. Press the key into the key aperture. Orient the key so that the T-shaped end fits into the rectangular groove on the connector. The other end of the key will be flush with the smooth side of the connector.

To remove a key from an AMP FDDI cable connector, turn the smooth side of the connector face up, place the tip of a small, flat-headed screwdriver on the key, and press firmly.

Preparation for Cabling Procedure

Be sure to keep the dust caps on the cable ends, transmitter(s), and receiver(s) until you actually make the connections. This prevents dirt and oils from getting on any important surfaces. Do not attempt to polish the connectors with a cloth made of synthetic fibers, as this will charge up the fiber and attract dust.

Run the fiber optic cable(s) out to reach the systems that will be connected. Be sure not to stretch, puncture, or crush the cable(s) with staples or heavy equipment or doors, etc. Always maintain the minimum bend radii, as specified by the cable manufacturer. The fiber in fiber optic cable can suffer damage if the cables are bent into small radii. The minimum bend radii is usually 10-20 times a cable's outer diameter.

WARNING

Never look into an active fiber optic cable. Harmful optical **RADIATION** capable of causing permanent eye damage may be present. If necessary, use a fiber optic power meter to determine whether a signal is present.

(This page is intentionally left blank to keep Figure 2-5, "Cabling a Single-MAC, Single-Attachment Station" with the corresponding cabling procedure.)

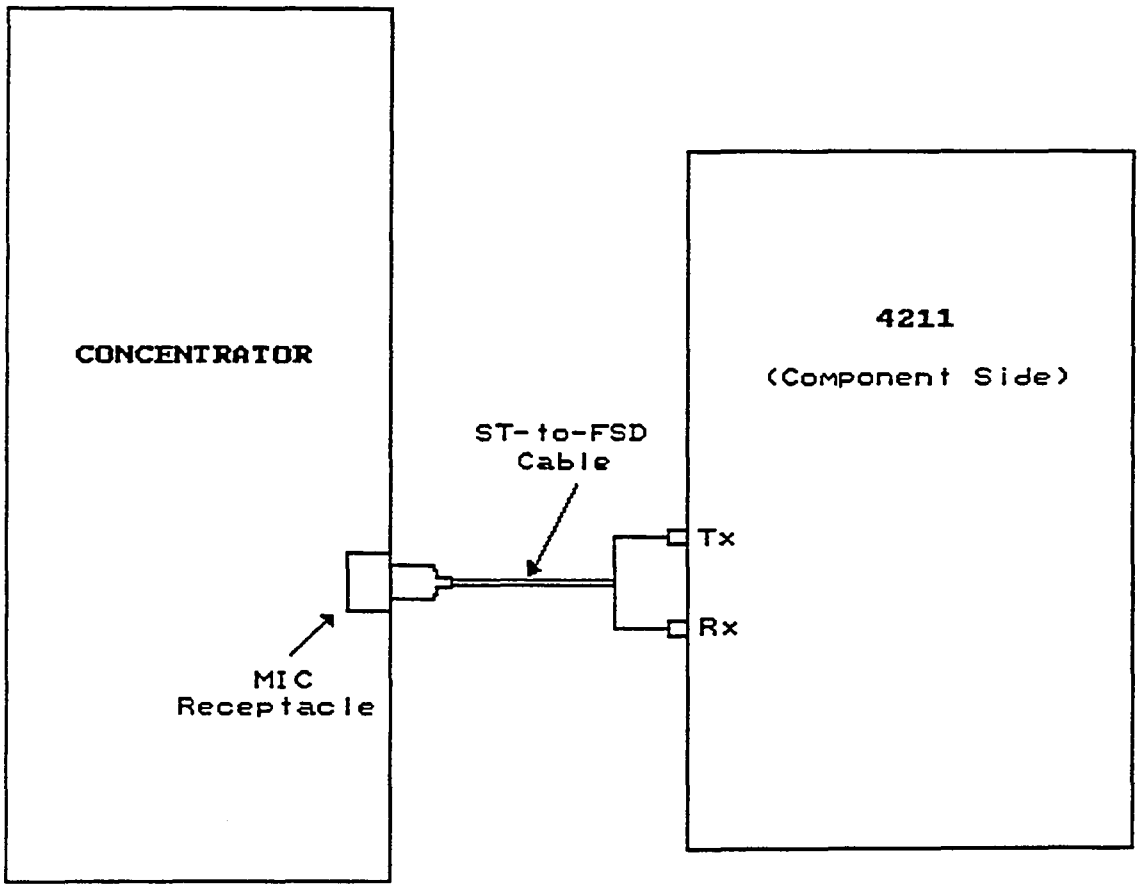


Figure 2–5 . Cabling a Single-MAC, Single-Attachment Station

Cabling a Single-Attachment Station

This subsection describes one method for cabling the 4211 as a single-attachment station. It assumes that you are using the sample FDDI cables and connectors listed in Table 2-3. The only sample parts required are:

- ST-to-MIC Cable with M keying
- MIC receptacle with M keying

The cabling procedure is described below. Refer to the figure on p. 2-14 for a drawing of the setup.

1. Connect the ST bayonet end of the ST-to-MIC cable to the transmit and receive connectors on the 4211.

   a) Locate the Tx and Rx connectors on the 4211. The single-PHY version of the 4211 has a single Tx/Rx connector pair. In the board layout on page 2-2, this is the pair of Tx/Rx connectors located closest to the J7 connector (optical bypass control).

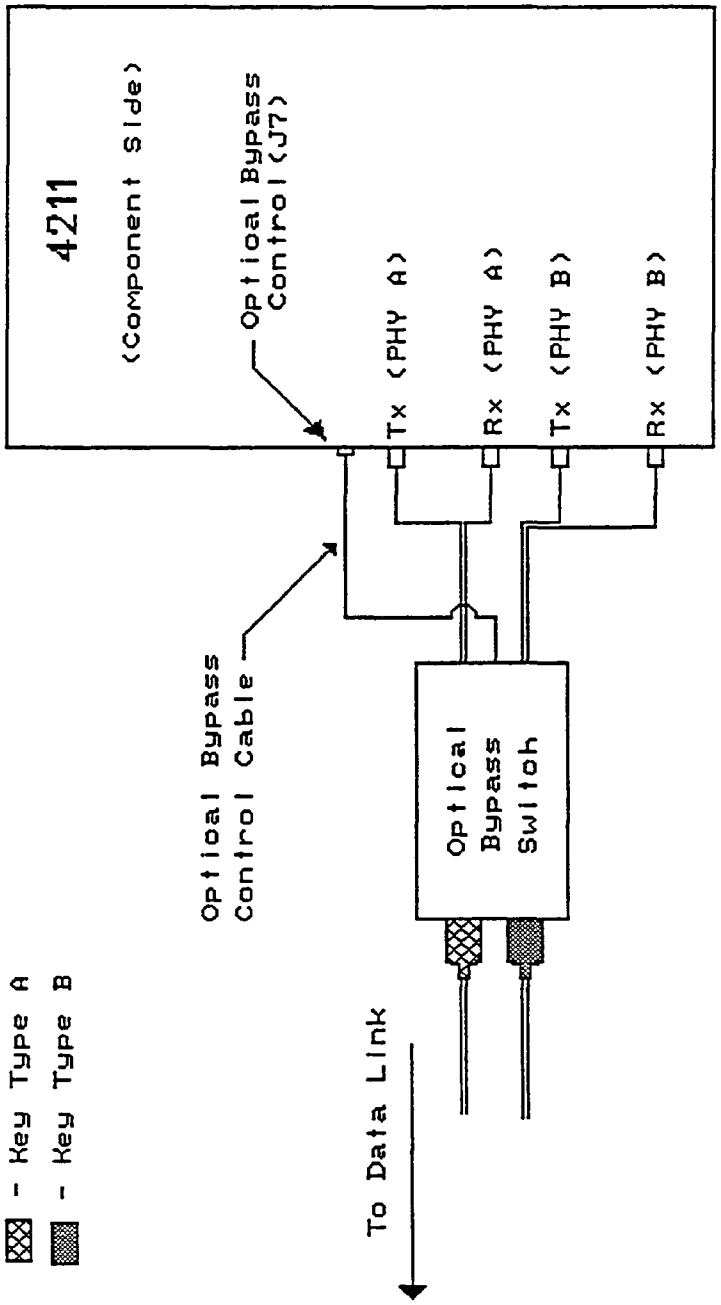
   b) Identify which ST connector should be plugged into the Tx connector, and which should be plugged into the Rx connector. Signals must be routed such that the Tx output of one station is routed to the Rx input of its downstream neighbor.

      There is no physical difference between the ST connectors on the cable, but they are usually marked in some way to distinguish them. These marks should be used consistently to cable all the stations in your installation. For example, if you decide to use the ST connector marked with a blue ring to route Tx signals from this station, you should do so for all other stations as well.

   c) The 4211's transmitter and receiver each has a gold jack with a slit running from the edge to its base. Key the ST connectors to the appropriate jack and push them on. Turn the connectors clockwise to lock the bayonet mechanism.

2. Connect the FDDI end of the ST-to-MIC cable to a MIC receptacle. This adapter is typically installed inside the concentrator.

3. Recheck your connections. In particular, make sure that each bayonet connector is plugged into the correct Tx or Rx connector as discussed in Step 1b.



NOTE: The above drawing assumes that the Tx/Rx connector pair closest to the J7 connector on the board is used for PHY A and the other Tx/Rx connector pair is used for PHY B.

Figure 2–6 . Cabling a Single-MAC, Dual-Attachment Station

Cabling a Single-MAC, Dual-Attachment Station

This subsection describes one method for cabling the 4211 as a single-MAC, dual-attachment station. It assumes that:

- You are using the sample FDDI cables and connectors listed in Table 2-3 (p. 2-9). The sample parts required are:
  - 1 Optical Bypass Switch
  - 2 MIC-to-MIC Cables, one with "type A" keying on one end and the other with "type B" keying on one end [1]
- The Tx/Rx connector pair closest to the J7 connector (optical bypass control) will be used for PHY A. The other Tx/Rx connector pair will be used for PHY B. This coincides with the way the Tx/Rx connectors are labelled in the board layout on p. 2-2. See "Considerations for Cabling a Dual-Attachment Station", p. 2-10, for additional information.

The procedure for cabling a single-MAC, dual-attachment station is as follows.

1. Connect the optical bypass switch to the 4211. To do so:

   a) Locate the 4211's Tx/Rx connector pairs for PHY A and PHY B, referring to Figure 2-6. Also see the board layout on p. 2-2, if necessary.

   b) Plug the ST connectors on the bypass switch into the appropriate Tx or Rx connector on the 4211. **Each ST connector MUST be plugged into the correct Tx or Rx connector** on the 4211. Signals must be routed such that each station's PHY A Tx output feeds to the PHY B Rx input of its downstream neighbor. Likewise, each station's PHY B Tx output must be routed to the PHY A Rx input of its downstream neighbor.

      To identify the connectors, examine the two strand-pairs of ST connectors on the optical bypass switch. They should be marked in some way by the cable manufacturer (e.g. with a number or color coding) to help you tell them apart.

      Assuming that the connectors are labelled "1", "2", "3", and "4", as shown in Figure 2-7, the following connections should be made:

[1] These cables connect the optical bypass switch to the rest of the network. Keying is only specified for end of the cable which connects to the bypass switch, since the connector requirements at the other end can vary.

ST Connector # on Bypass Switch	Mates to:	ST Connector on 4211
"1"		PHY B Tx (Primary Tx)
"2"		PHY B Rx (Secondary Rx)
"3"		PHY A Tx (Secondary Tx)
"4"		PHY A Rx (Primary Rx)

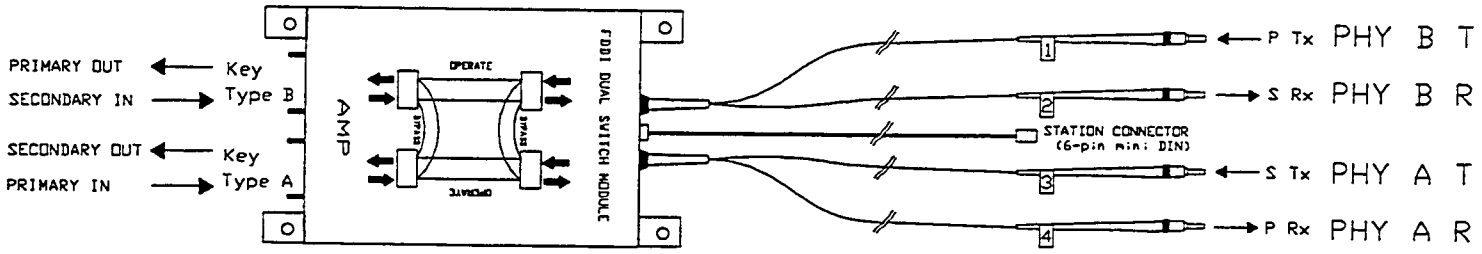


Figure 2–7 . Example Optical Bypass Switch

c) Pull the dust caps off the transmitters, receivers, and bayonet connectors.

d) Notice that each transmitter and receiver each has a gold jack with a slit running from the edge to its base. Key the bayonet connectors to the appropriate jack and push them on. Turn the connectors clockwise to lock the bayonet mechanism.

e) The bypass switch has a control cable with a 6-pin mini DIN connector on the end. Plug this connector into the optical bypass control connector (J7) on the 4211. To locate the J7 connector, refer to Figure 2-6 or the board layout on p. 2-2.

2. Connect the optical bypass switch to the data link. To do so:

a) Locate the two MIC receptacles on the optical bypass switch. These are on the end of the switch opposite from the cables discussed in Step 1. One socket has "type A" keying, and the other has "type B".

b) Using the MIC-to-MIC cable with "type A" keying on one end, connect the socket with "A type" keying to the data link.

c) Using the MIC-to-MIC cable with "type B" keying on one end, connect the socket with "B type" keying to the data link.

3. Recheck all your connections. In particular, make sure that each bayonet connector is plugged into the correct Tx or Rx ST connector.

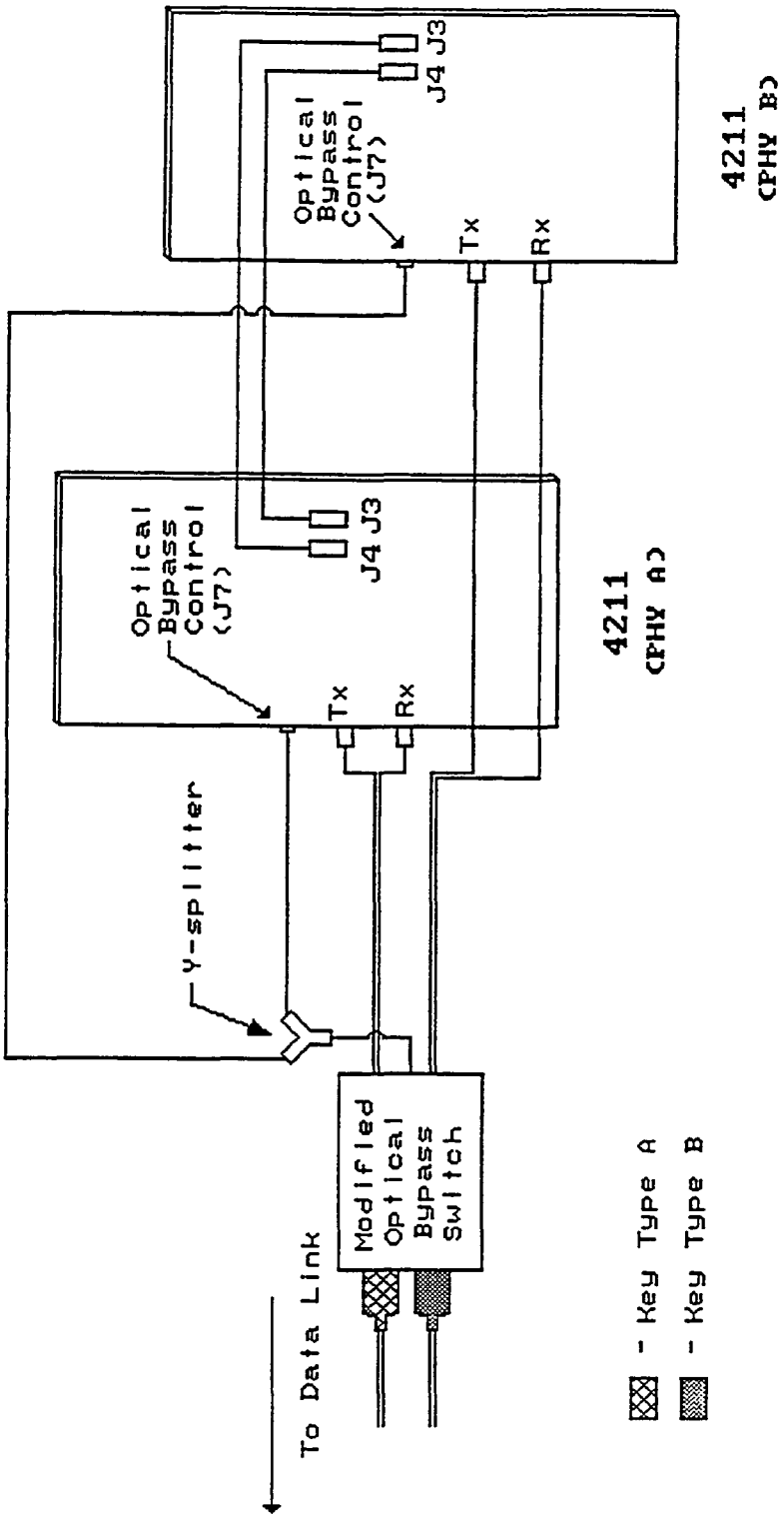


Figure 2–8 . Cabling a Dual-MAC, Dual-Attachment Station

Cabling a Dual-MAC, Dual-Attachment Station

This subsection describes one method for cabling two 4211s as a dual-MAC, dual-attachment station. It assumes that you are using the sample FDDI cables and connectors listed in Table 2-3 (p. 2-9). The sample parts required are:

- 2 Dual MAC Cables
- 1 Modified Optical Bypass Switch[2]
- 2 MIC-to-MIC Cables, one with "type A" keying on one end and the other with "type B" keying on one end[3]

The dual-MAC cabling procedure differs from single-MAC in that part of the cabling procedure takes place **before** inserting the boards into the VMEbus chassis. The rest of the cabling is done **after** the boards are installed.

Also note that the two 4211s used in a dual-MAC, dual-attachment setup only have one transmitter/receiver pair each. The user decides which of the two boards will be used for PHY A, and which for PHY B.

The cabling procedure is described below and depicted in Figure 2-8.

1. Locate the J3 and J4 connectors on each of the two 4211s. On the side of the board closest to these connectors, a small section of the board is perforated to allow it to be easily broken off. This breakaway section is specifically intended to accommodate dual MAC configurations. If it has not already been removed at the factory, use a pair of pliers to carefully break it off.

2. Hold the two 4211s parallel, a few inches apart. Align them so that:
   — the component side of each board is facing the same direction
   - and -
   — the VMEbus connectors P1 and P2 on each board are facing the same direction

   Using a Dual MAC Cable,[4] connect the J3 connector on one board to the J4 connector on the other. Use a second Dual MAC Cable to interconnect the other pair of J3 and J4 connectors. The cables should wrap around the edge of one 4211 to the other, fitting snugly into the breakaway section discussed in Step 1.

3. Holding the interconnected boards parallel, slide them into adjacent VMEbus card slots. Once they are properly seated in the slots, tighten the captive mounting screws on the board front panels.

4. Decide which of the two 4211s will be used for **PHY A** and which for **PHY B**.

---

[2] To implement the optical bypassing function on two boards, the user must modify the optical bypass switch so that signals from its control cable are routed to both boards. This could be done using a Y-splitter.

[3] These cables connect the optical bypass switch to the rest of the network. Keying is only specified for end of the cable which connects to the bypass switch, since the connector requirements at the other end can vary.

[4] A 3" flat-ribbon cable described in the sample parts list on p. 2-9.

5. Connect the Modified Optical Bypass Switch to the boards as follows:

   a) Plug the ST connectors on the bypass switch into the appropriate Tx or Rx connector on the 4211. **Each ST connector MUST be plugged into the correct Tx or Rx connector** on the 4211. Signals must be routed such that each station's PHY A Tx output feeds to the PHY B Rx input of its downstream neighbor. Likewise, each station's PHY B Tx output must be routed to the PHY A Rx input of its downstream neighbor.

   To identify the connectors, examine the two strand-pairs of ST connectors on the optical bypass switch. They should be marked in some way by the cable manufacturer (e.g. with a number or color coding) to help you tell them apart.

   Assuming that the connectors are labelled "1", "2", "3", and "4", as shown in Figure 2-9, the following connections should be made:

ST Connector # on Bypass Switch	Mates to:	ST Connector on 4211
"1"		PHY B Tx (Primary Tx)
"2"		PHY B Rx (Secondary Rx)
"3"		PHY A Tx (Secondary Tx)
"4"		PHY A Rx (Primary Rx)

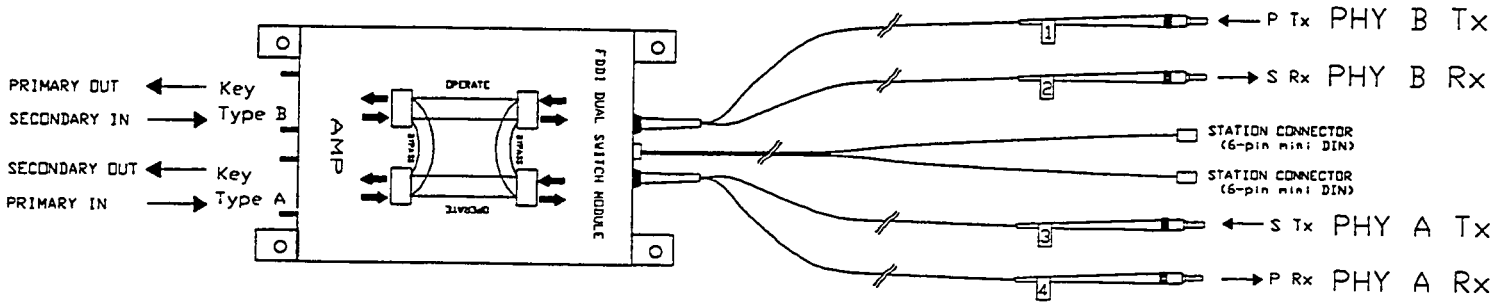


Figure 2-9 . Example Modified Optical Bypass Switch

   b) Pull the dust caps off the transmitters, receivers, and bayonet connectors.

   c) Notice that each ST transmitter and receiver each has a gold jack with a slit running from the edge to its base. Key the bayonet connectors to the

appropriate jack and push them on. Turn the connectors clockwise to lock the bayonet mechanism.

d) The control cable on the modified bypass switch has its signals split over two cables. Plug these cables into the optical bypass control connectors (J7) on the 4211s. To locate the J7 connector, refer to Figure 2-8 or the board layout on p. 2-2.

4. Connect the optical bypass switch to the data link.

   a) Locate the two FDDI sockets on the optical bypass switch. These are on the end of the switch opposite from the cables discussed in Step 3. One socket has "type A" keying, and the other has "type B".

   b) Using the MIC-to-MIC cable with "type A" keying on one end, connect the socket with "A type" keying to the data link.

   c) Using the MIC-to-MIC cable with "type B" keying on one end, connect the socket with "B type" keying to the data link.

5. Recheck all your connections. In particular, make sure that each bayonet connector is plugged into the correct Tx or Rx connector.

Step 6. Power System On

Once the 4211 is installed in the chassis and properly cabled, turn the host on. **If your other system documentation (host/peripherals) contains precautionary statements about powering up the system, please be sure to follow these precautions.**

After powering up the system, observe LED1 and LED2, which are located on the edge of the board opposite the VMEbus connectors. The red LED (LED1) and green LED (LED2) will successively toggle on and off at regular intervals for about ten seconds. This indicates that the 4211 is performing its power-up diagnostics.

Once the power-up diagnostics complete successfully, the LEDs will stop toggling on/off and the green LED will stay on. At this point, the 4211's Common Boot Interface is booted up and ready to accept commands. (See "Bootup Sequence", p. 3-3). For more information on the LEDs, see p. 5-3.

For more information on power-up diagnostics, see p. 6-1.

NOTES: [1] Once the host boots the RC Interface, the LEDs return to toggling on/off.

[2] If one of the power-up tests fails, the LEDs will start repeating a pattern of on/off signals of varying duration. Likewise, if both LEDs turn on at the same time or the green LED fails to turn on, the onboard 29000 processor has failed. If either of these events occur, contact Customer Service at Interphase for assistance.

[3] If *both* LEDs fail to turn on, the 4211 may not be fully seated in the chassis. Turn off the system and ensure that the 4211 is firmly inserted into the backplane slot.

If the LEDs still do not light up after reapplying power to the system, call Customer Service at Interphase for assistance.

CHAPTER 3
FUNCTIONAL OVERVIEW

OVERVIEW OF INTERFACE

The host issues commands to the 4211 via an intelligent interface called the Report/Command (RC) Interface. This interface is implemented in the 4211's 512-byte short I/O space. The term "short I/O" refers to a block of onboard memory configured to operate in the VMEbus Short Supervisory I/O address space. This memory is shared across the bus between the host CPU and the 4211's firmware.

The 4211's 512-byte short I/O space may be viewed as a "window" into the 1-megabyte address range controlled by the onboard 29000, as depicted in Figure 3-1.

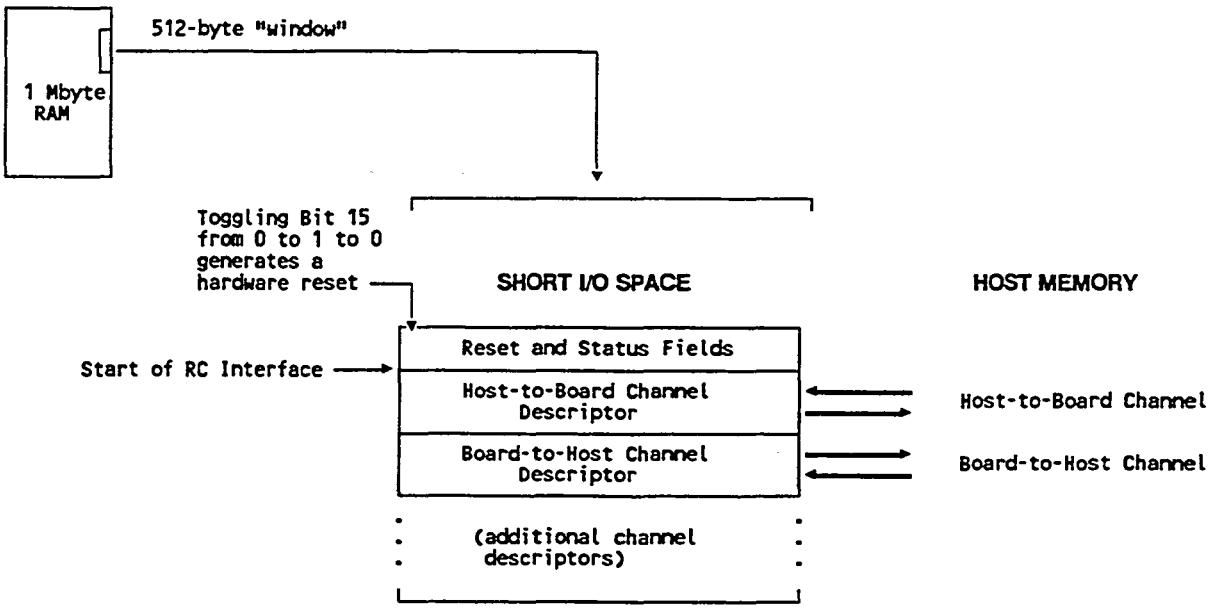


Figure 3–1 . Short I/O as a Window into the 4211's 1-Megabyte Address Space

SYSTEM REQUIREMENTS

To interface with the 4211, the host must meet following requirements:

1. The 4211's short I/O memory is configured for **16–bit read/write access only.**

> **IMPORTANT !**
> The 4211 does **NOT** support 8- or 32-bit accesses to its short I/O space.

2. To transfer a 32-bit quantity to/from short I/O, the host must place the MSW (Most Significant Word) in the low-order address and the LSW (Least Significant Word) in the high address.

3. Host-to-board segments must be longword-aligned. (For details, see "Managing Segment Memory" in the *Common Boot & RC Interface User's Guide*.)

4. At least 512 bytes of VMEbus short I/O address space must be available on the system VMEbus for use by the RC Interface.

5. The byte ordering of commands must be "big-endian" (i.e. Most Significant Byte first).

For more information on what the 4211's hardware does (and does not) support, refer to Chapter 5.

For details on the capabilities and limitations of the 4211's firmware, read this chapter. Later chapters describe specific aspects of the firmware when they are relevant to the topic being covered.

A Note on Other Prerequisites

Before attempting to write or modify a driver for the 4211, you need an extensive technical background. As noted at the beginning of this manual, prerequisite skills include knowledge of FDDI, C, VMEbus, and the host's operating system. The level of expertise required in each of these areas will naturally depend on your design task.

Listed below are several aspects of the 4211's firmware, along with comments on what you need to know to use each feature.

1. **Basic Interfacing.** In order to issue commands to and read responses from the 4211, you must understand the Report/Command (RC) Interface. This interface is documented in the *Common Boot and RC Interface User's Guide*.

2. **Network Operations (Default SMT).** Performing network operations using the 4211's default SMT setup requires a good understanding of FDDI, especially the SMT component. You need to know how the Management Information Base (MIB) works, as well as how to use the many parameters (MIB attributes) stored in the MIB.

3. **Customizing the SMT.** The 4211's SMT can be modified by assigning new values to the MIB attributes (excluding attributes which are read-only). This requires EXPERT-LEVEL knowledge of SMT, the MIB, and MIB attributes. See "Setting MIB Attributes", p. 3-15, for more information.

SHORT I/O AND THE RC INTERFACE

Short I/O memory is initially used for the purposes of booting the controller and starting up the RC Interface. Once the RC Interface is up and running, short I/O will hold **channel descriptors**. These descriptors point to areas of host memory containing RC commands and responses, which in turn point to transmit/receive data elsewhere in memory. They act as control elements of a low-overhead handshake mechanism for queuing commands asynchronously between the controller and the host.

Channels are data structures used for host-to-board or board-to-host communication. Commands are written to the board through a **host–to–board channel**. If an issued command needs to report

command completion status to the host, it does so through a **board-to-host channel**. Channels are discussed at length in Chapter 3 of the *Common Boot and RC Interface User's Guide.*

For a detailed memory map of the 4211's short I/O space, refer to the figure "Control Structure of RC Interface" in Chapter 3 of the *Common Boot and RC Interface User's Guide.*

COMMON BOOT INTERFACE

The Common Boot interface provides a boot and initialization mechanism for the 4211. The host uses Common Boot to submit the characteristics of the initial RC channel pair needed to boot the RC Interface on the 4211. This initial channel pair can then be used to create additional channels after RC is up and running.

Once RC boots successfully, the Common Boot interface exits until the board is reset.

The host can also perform a variety of other tasks when in Common Boot mode. These include:

- performing extended diagnostics (see Chapter 6)
- downloading code for onboard execution [5] (see Appendix D)
- manipulating the LEDs, peeking/poking memory, and other functions (see Chapter 2 of the *Common Boot and RC Interface User's Guide*)

The 4211 supports all commands in the Common Boot command set.

Bootup Sequence

The following sequence of events occurs each time the 4211 is powered up or reset

1. The 4211 executes its bootstrap code and performs a series of power-up diagnostics. While the 4211 performs the tests, LED1 (a red LED) and LED2 (a green LED) toggle on and off. Then, once the 4211's firmware has booted up and the Common Boot interface is active, LED1 turns off and LED2 turns on and remains lit.

2. The Common Boot Interface starts up and fills the 512-byte short I/O space with a 32-bit value called CB_HERALD. The format of CB_HERALD is shown in Figure 3-2.

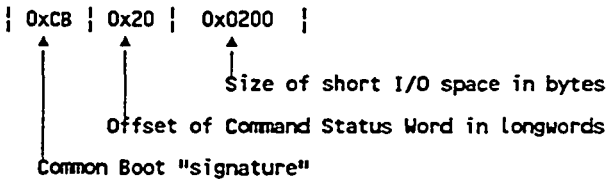


Figure 3-2 . CB_HERALD Format

[5] This feature is for special applications only. It is not needed to boot the native RC Interface stored in controller firmware.

The offset in CB_HERALD points to the location in short I/O containing the Common Boot Command Status Word (+0x20 longwords or 0x80 bytes). The length field in CB_HERALD indicates how much short I/O space is available (512 bytes).

3. The host reads Command Status Word. It should contain one of the following 32-bit ASCII values: "CBOK" (0x43424F4B) or "FAIL" (0x4641494C). If it contains CBOK, the host may issue any Common Boot command. If it contains FAIL, the host may issue the DIAG command to identify the problem.

   For more information on DIAG, see the chapter on the 4211's diagnostics (p. 6-1). For a description of the Common Boot command set and the procedure for issuing commands, refer to the *Common Boot and RC Interface User's Guide*.

4. The board polls the Command Status Word. When it changes to a command code, the board executes the command and writes the results (CBOK or FAIL) back to the Command Status Word. It then polls for the next command.

   No interrupt is generated at the completion of a Common Boot command, so the host also polls the Command Status Word to determine whether the command has completed.

5. To boot the RC Interface, the host issues the Common Boot command "BOOT". Once the the RC Interface boots up, LED1 and LED2 return to toggling on/off.

6. The RC Interface starts up and creates the initial host-to-board and board-to-host channel. Common Boot exits until the board is reset.

RESETTING THE 4211

To reset the 4211, toggle bit 15 of the Hardware Control field from 0 to 1 to 0. The Hardware Control field is a 16-bit field located at the start of short I/O (i.e. bytes 0 and 1 of the 512-byte short I/O space). When the 4211 is reset, it goes through its bootup sequence as described in the previous section.

NOTE: After toggling the "reset bit", the host should wait at least 500 ms before looking for CB_HERALD.

INTERRUPTS

The host enables interrupts from the 4211 on a per-channel basis. This is done by issuing separate Set Interrupt directives for the appropriate board-to-host channels. Such channels will cause the 4211 to interrupt the host when the necessary internal conditions are met.

Like the host, the 4211 writes to its board-to-host channels asynchronously. New messages may come from the 4211 at any time. The host need not worry about synchronization, however. It merely processes the messages between the channel descriptor's read pointer and write pointer and then updates the board-to-host read pointer.

For more information, see the section on interrupts in the *Common Boot and RC Interface User's Guide*.

ISSUING COMMANDS

Once it has booted the RC Interface, the host can issue commands to the 4211 through the initial host-to-board channel and read 4211-generated messages from the initial board-to-host channel. The *Common Boot and RC Interface User's Guide* describes the procedures for doing so.

Available RC Commands

The 4211 FDDI command set consists of those RC commands which are *specific* to the 4211. 4211-specific commands are described in the next chapter.

In addition to the FDDI command set, the 4211 supports all members of the generic RC command set. The generic RC command set is documented in the *Common Boot and RC Interface User's Guide.*

Before you can use the 4211 to perform FDDI-related tasks, it is necessary to have a good understanding of the RC Interface. Concepts such as "segment linking" and "working one's way through a channel as the writer (or reader) of a channel" are integral to writing a driver for the 4211. For a description of how the RC Interface works, see chapter 3 of the *Common Boot and RC Interface User's Guide.*

Location of Commands and Tx/Rx Data

The 4211 expects queued commands as well as transmit/receive data to reside in host memory. It does not implement a "master mode" memory access that would allow it to access global VMEbus memory directly. Instead, it transfers commands, responses, and transmit/receive data between the host and controller-resident buffers using a high-speed, on-board DMA channel.

CONNECTING THE STATION TO THE RING

The host instructs the 4211 to connect to (or disconnect from) the FDDI ring by issuing a Ring Control directive. If the command completes successfully, the 4211 issues a Ring Status Indication to report the new ring status — ring "up" or ring "down".

Be aware that the 4211 only issues a Ring Status Indication **when the ring status changes.** If the 4211 fails in its attempt to connect (or disconnect), it does not issue the indication. Likewise, it does not issue the indication if it receives a directive to connect/disconnect when it is already in the specified state.

Assuming that the station is not currently connected to the ring, the sequence of events for a successful connection attempt is as follows:

Step 1 — The host issues a Ring Control directive with the value 0x1 (Connect to Ring) in the options field. See p. 4-20 for a description of this directive.

Step 2 — After connecting to the ring, the 4211 issues a Ring Status Indication with the value 0x1 (Ring Up) in the status field. This indication is written to the default response channel.[6] See p. 4-52 for a command description.

Step 2 should occur a few seconds after Step 1. If it does, the station has successfully connected to the ring and the host can proceed to transmit and receive frames.

If Step 2 does NOT occur within 5 - 10 seconds, the station has failed in its attempt to connect to the FDDI ring. The connection attempt will succeed only if a wide range of parameters are properly set for the ring. These requirements, which involve multiple layers of the network interface, are documented in the FDDI SMT specifications.

Additional information about the state of the ring is signalled by LED3, LED4, and LED5. To locate these LEDs, refer to the board drawing on p. 2-2. See p. 5-3 for a description of the LEDs.

[6] The default response channel is the initial board-to-host channel created by the host when it boots the RC Interface. The host can detect pending messages by polling the channel and/or enabling interrupts, as discussed at length in Chapter 3 of the Common Boot and RC Interface User's Guide.

TRANSMITTING DATA

In order to transmit a frame over the network, the host writes a transmit command (either Transmit Raw Data or Transmit Datagram) to a host-to-board channel. The command points to the outgoing data somewhere in host memory. If desired, the frame may be stored as fragments in discontiguous memory locations (see "Scatter/Gather Capabilities" p. 3-12).

For details on the two data transmission commands, refer to pages 4-6 and 4-10, respectively.

Queuing Frames for Transmission

When the 4211 processes a transmit command, it: 1) attempts to move the data to be transmitted into its one-megabyte communications buffer, and 2) issues a Tx Response (p. 4-45). This response tells the host either that: 1) the data has been successfully transmitted over the network, or 2) an error has occurred.

Once the host has received the Tx Response, it can reuse the memory space from which the 4211 obtained the frame.

The 4211 queues outgoing frames in its communications buffer until it gets the token. It then transmits as many frames as possible until the token holding time in the MAC expires. Any untransmitted frames are held until the 4211 gets the token back.

Synchronous vs. Asynchronous Frames

In general, all queued synchronous frames are transmitted before asynchronous frames. Note, however, that the RC Interface does not guarantee that a synchronous frame queued for transmission in a host-to-board channel will go out before any asynchronous frames queued in the channel. RC processes commands in the channel in a first-come first-served manner. As the 4211 executes each Tx command, it pulls the corresponding frame across the bus. At this point, it still has not determined which frames should go first.

For example, suppose the host queues up ten asynchronous frames followed by a synchronous frame in the same host-to-board channel. Each frame is transferred onboard and queued internally at the FORMAC level.

If the token has NOT been captured at this time, the synchronous frame will be transmitted before the ten asynchronous ones once the 4211 gets the token.

On the other hand, if 4211 captures the token at the same time that one or more synchronous frames are being queued on board, any asynchronous frames which are already queued up may be transmitted before the synchronous one(s).

Two DMA status LEDs — LED6 and LED7 — respectively signal the transmission of synchronous and asynchronous frames (see p. 5-3 for details).

RECEIVING DATA

When data comes in from the network for the host, the 4211 stores it internally in its communications buffer and uploads it as soon as possible to the host. Two DMA status LEDs — LED8 and LED9 — signal the readiness of channels A and B to receive data (see p. 5-3 for details).

In order for the 4211 upload received frames, the host must supply the controller with a **buffer list** consisting of **buffer list elements** which point to **receive buffers** in host memory. These terms have a meaning specific to the RC Interface and are defined in the following subsections (pp. 3-8 to 3-12).

To upload a received frame, the controller does the following:

1) The 4211 DMA's the frame to a host-resident receive buffer. If the frame is larger than the buffer, the 4211 scatters the frame across multiple buffers (see "Scatter/Gather Capabilities", p. 3-12, for details on frame scattering).

   Only one frame (or frame fragment) is uploaded per buffer. If no receive buffers are available, the 4211 holds the frame internally until it can upload it.

2) The 4211 then issues a **receive indication** which points to the receive buffer(s) containing the uploaded frame. The 4211 command set currently supports one receive indication — Rx Frame. See p. 4-47 for a command description.

   All receive indications are posted to the default response channel (i.e. the initial board-to-host channel created at RC boot time).

   Note that receive indications are written to the default response channel, but the actual received frames are not. The host must separately allocate memory space for these two purposes. The considerations involved in allocating memory for RC channels (i.e. segment memory) are documented in the *Common Boot and RC Interface User's Guide*. Guidelines for setting up receive buffers are discussed below (pp. 3-8 to 3-12).

Alignment of Received Frames

When the 4211 uploads a frame to a receive buffer, it aligns the information field of the frame on a longword boundary. To do so, it places a 3-byte pad preceding the MAC frame header.[7] This 3-byte pad is *not* included in the "Frame Length" field of the receive indication.

Allocating and Managing Receive Buffers

To allocate host memory for incoming frames, the host issues the Set Up Receive Buffers directive (p. 4-14). This directive is issued at initialization time. It defines a **buffer list** which points to **receive buffers** — a set of same-sized, longword-aligned buffers to which the 4211 may upload received frames.

---

[7] The MAC frame header is 13 bytes, including the Frame Control field (1 byte) and the Source and Destination addresses (6 bytes each).

Creating a Buffer List

To set up a buffer list, the host allocates a block of contiguous memory (usually a page). It then writes **buffer list elements** into this space, terminated by the value 0xFFFFFFFF.

Each buffer list element points to a specific receive buffer somewhere in host memory. All receive buffers must be of the same size and longword-aligned. Valid buffer sizes range from 512 bytes to 8K bytes, inclusive.

Format of Buffer List Element. Figure 3-3 shows the format of a buffer list element. Its data structure is shown on p. B-13.

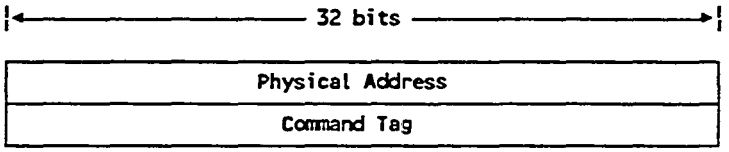


Figure 3–3 . Buffer List Element

The "Physical Address" field is used by the 4211 to DMA data to the buffer. As noted previously, receive buffers must be aligned on a longword boundary.

The "Command Tag" field is a value assigned by the host which will be echoed back by the controller via a receive indication. This tag is usually the address of a host memory descriptor or virtual memory address for the buffer.

Marking the Last Valid Element in the Buffer List. The host does not have to fill the buffer list completely with valid elements (that is, elements which point to an available receive buffer). Instead, it maintains a **delimiter** located immediately after the last valid element. The delimiter has the same format as a buffer list element, but has the special value 0xFFFFFFFF in its Physical Address field. Thus, if you wish to create a buffer list containing N elements, then the number of elements specified using the Set Up Receive Buffers directive should be N + 1 to allow for the delimiter.

Ownership of Receive Buffers

Each valid element in the buffer list points to a receive buffer in host memory. The 4211 "owns" the receive buffer until it has placed data in it and notified the host by issuing a receive indication. At that time, ownership of the buffer returns to the host. The host must therefore post new buffers to the list to keep the 4211 supplied with available receive buffers. See "Host-Managed Tasks", p. 3-11, for details.

Managing the Buffer List

Board-Managed Tasks. When the 4211 executes the Set Up Receive Buffers directive, it DMAs a copy of the entire buffer list onboard. As it processes received frames coming off the net, it starts working its way down the list, using successive buffer list elements to identify receive buffers to which it can upload frames.

Each time it uploads a frame to one or more buffers, the 4211 moves its list pointer ahead to the next available buffer list element. It then issues a receive indication to notify the host that: 1) received data is ready for the host to process, and 2) one or more buffers can be reused (or newly allocated) and posted to the buffer list. The method for posting buffers to the list is described in "Host-Managed Tasks", below.

If the buffer size is 4.5K bytes or larger, there will always be a one-to-one correspondence between the number of posted buffers and the number of received frames which have been uploaded to the host. This is because the 4211 only places one frame (or frame fragment) in each buffer. If a smaller buffer size is being used, large frames will be scattered over multiple buffers. The buffer size is defined using the "Length in Bytes of Each Buffer" field in the Set Up Receive Buffers directive (p. 4-14).

The 4211 continues moving down the buffer list until one of the following conditions occurs:

1) It reaches the physical end of the list as defined by the Set Up Receive Buffers directive. In this case, the 4211 simply wraps back to the top of the list and continues to work down the list.

   - OR -

2) It runs out of valid elements (i.e. its list pointer reaches the delimiter). In this event, it re-reads the host's copy of the buffer list to obtain new buffers that the host has posted to the list. If the host has not updated the list, the 4211 polls the list approximately every 32 ms. until it determines that the delimiter has moved. It then refreshes its onboard copy of the buffer list and goes back to uploading frames.

The host can reduce the number of times that condition #2 occurs by placing an appropriate value in the "Rx Load Adjustment" field of the Set Up Receive Buffers directive. This field causes the 4211 to refresh its copy of the buffer list more frequently during periods of high network traffic. For details on this feature, see "Balancing the Load of Rx Frame Processing", p. 3-11.

Each time the 4211 gets the buffer list from the host, it wipes out its old copy of the buffer list. However, it does not return its list pointer to the top of the list. Instead, it continues to point to the same buffer list element that it was pointing to before it re-read the list. If that specific element in the refreshed list *is not* the delimiter, the 4211 assumes that it is a valid element and proceeds to upload data as usual. On the other hand, if that element *is* the delimiter, the 4211 assumes there are no free buffers for uploading frames. It will poll the buffer list until the host posts new buffers to it.

Host-Managed Tasks. To manage the buffer list, the host typically keeps a reader pointer and writer pointer into the list. The reader pointer points to the "next" buffer list element from which the host expects to obtain data. The writer pointer points to the delimiter.

As the 4211 passes receive indications back to the host, the host extracts the data from the buffers and moves the reader pointer down the list to the next free buffer.

To post a buffer back to the list, the host should:

1) Write the delimiter (0xFFFFFFFF) into the Physical Address field of the buffer list element immediately after the old delimiter. (At this point, the list contains *two* delimiters in consecutive buffer list elements.)

2) Insert the new buffer at the writer pointer, overwriting the old delimiter. Be sure to fill in the Command Tag field before the Physical Address field. This eliminates a window during which the 4211 could pick up the new physical address before the new command tag becomes valid.

3) Update the writer pointer to point to the new delimiter.

The above procedure is necessary to prevent the 4211 from inadvertently reading past the delimiter.

It is safe for the host to wrap to the top of the list so long as it posts a new buffer only *after* the 4211 has passed back one of "its" buffers via a receive indication.

Balancing the Load of Rx Frame Processing

On occasion, frames may arrive in large batches that cause inefficiencies in the way the 4211 and host process the data. For example, suppose that 64 frames addressed to the host arrive in a very small time frame. If the 4211 tries to process and upload all 64 frames at once, other important tasks (such as pending transmits) will be left hanging. Moreover, the buffer list is likely to become depleted. The 4211 will then have to wait for the host to process the uploaded data and post new buffers to the list before it can get buffers for additional frames.

This situation can be avoided by placing an appropriate value in the "Rx Load Adjustment" field of the Set Up Receive Buffers directive. This field sets the maximum number of received frames that the 4211 will process at a given time. It also causes the board to refresh its copy of the buffer list each time this maximum value is exceeded.

Using the previous example, assume that the 4211 has 64 frames to upload and the "Rx Load Adjustment" field is set to 16. The 4211 will upload 16 frames and then break out of the receive frame handling routine to re-read the buffer list. Other tasks, such as pending transmits, are attended to at this time. Then control is returned to the receive routine to process the next 16-frame chunk.

The optimum value for the "Rx Load Adjustment" field depends on a wide range of variables. These include the speed of the host processor, the size of the receive buffers, the amount of network traffic addressed to the host, and other factors. To find best

setting, test different values in the field. A good rule of thumb is to start with the smaller of the following two values:

- one-half the number of elements in the buffer list

  - OR -

- one-half the number of internal data buffers for received frames on the 4211 [8]

The Rx Load Adjustment feature is disabled if the host clears the field to 0 or enters an out-of-bounds value. If the feature is disabled, the receive frame handling routine will attempt to process all the received frames it knows about, even if it has to use up all available buffers in the buffer list. It also means that the 4211 will not refresh its copy of the buffer list until it encounters the delimiter (0xFFFFFFFF). For a statement of the values currently supported in the "Rx Load Adjustment" field, see p. 4-15.

Insufficient Receive Buffers

Since the 4211 relinquishes a buffer after it has placed received data in it and notified the host, it is the host's responsibility to post new buffers to the buffer list. If it fails to do so, the 4211 will eventually use up all of the buffers in the list. It then buffers frames internally until it runs out of available RAM. Subsequent incoming frames addressed to the host will be lost. No incoming data can be received until the host allocates new buffers to the list, thereby enabling the board to free up its onboard memory by uploading previously received data to the host.

SCATTER/GATHER CAPABILITIES

The 4211's scatter/gather feature enables it to:

1) Upload a received frame to discontiguous locations in host memory ("frame scattering")
2) Download a transmit frame from discontiguous locations in host memory ("frame gathering")

The considerations involved in frame scattering are different from those of frame gathering. This is because the rules for allocating and using receive buffers are much stricter than those concerning outgoing data.

Frame Scattering

If the size of a received frame exceeds that of the 4211's receive buffers, the 4211 will scatter the frame across multiple buffers. It then reports the location of the frame fragments in the receive indication. As discussed previously, the 4211 only writes one frame (or frame fragment) to a receive buffer before passing it back to the host.

[8] At the time of this writing (11/24/92), this value is 0x40.

Frame Gathering

If the host issues a transmit request containing multiple frame fragments, the 4211 will attempt to gather the frame. In order for this attempt to be successful, the host must observe several rules concerning the start and end addresses of the Tx frame fragments. These rules, which do NOT apply to contiguous Tx frames, are as follows:

1. The first Tx frame fragment can start on any byte boundary.
2. The starting address of rest of the fragments must be word (16 bit) aligned.
3. All other fragments must end on a word boundary, except for the last one. The last fragment can end on any byte boundary.

The above rules are depicted in Figure 3-4.

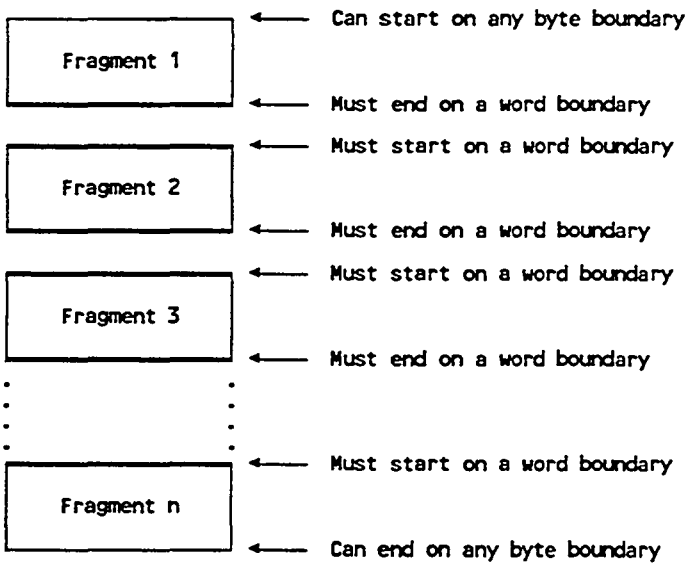


Figure 3-4. Proper Placement of Fragments in Host Memory to Allow Frame Gathering

The 4211 will be able to download the fragments using longword (32-bit) data transfers, ***provided that*** the following condition is met. The host must longword-align all the frame fragments, with two permitted exceptions. The exceptions are: 1) the start address of the first fragment, and 2) and the end address of the last fragment.

If the host does not longword-align the fragments as described above, the 4211 will download some or all of the fragments using 16-bit transfers (even if the Transfer Options Word specifies 32-bit transfers). If the host specifies a 32-bit block transfer and the 4211 is obliged to switch to 16 bits, the board will perform 16-bit block transfers. The 4211 will not perform 8-bit data transfers under any circumstances.

ACCESSING THE CAM

The 4211 includes a 256-entry CAM (content-addressable memory) in which the host can store up to 256 destination addresses. The host can add and remove addresses from the CAM, read specific entries in the CAM, and manipulate certain functions in the CAM Command Register. These tasks are performed using certain commands in the 4211's controller-specific command set, as documented in the next chapter. These commands are:

- Set CAM Address (p. 4-31) - add an entry to the CAM
- Clear CAM Address (p. 4-33) - remove an entry from the CAM
- CAM Address Response (p. 4-54) - reports the success/failure of a Set CAM Address or Clear CAM Address request
- Flush CAM Addresses (p. 4-34) - remove all entries from the CAM
- Get CAM Address (p. 4-43) - read an entry in the CAM
- Report CAM Address (p. 4-64) - reports the entry requested by a Get CAM Address request
- Set Hardware Parameters (p. 4-37) - change bits in the CAM Command Register
- Get Hardware Parameters (p. 4-41) - query the CAM Command Register
- Report Hardware Parameters (p. 4-59) - reports the current value of the CAM Command Register, if requested to do so by Get Hardware Parameters

NOTE: Be careful to distinguish between Report CAM Address and CAM Address Response. The former reports the current address stored in a CAM entry. The latter is issued in response to an attempt by the host to add or remove a CAM entry.

In addition to the above commands, the 4211 supports a diagnostics command - DIAG - which can be used to test the CAM. Unlike the commands discussed above, DIAG is issued via the Common Boot interface instead of the RC Interface. See the chapter on diagnostics for details, especially p. 6-16.

ACCESSING THE MIB

Since SMT functions[9] are fully implemented in the 4211's firmware, the Management Information Base (MIB) is also stored onboard. The MIB contains a large set of parameters (MIB attributes) which are used by the 4211 when performing SMT-level tasks. Appendix B includes C defines of the MIB attributes, starting on p. B-20.

The default MIB parameters on the 4211 are correct for most installations. In general, they should be left in their default settings.

The 4211 supports commands for reading or changing the value of MIB attributes. These commands, which are documented in the next chapter, may be issued any time after the RC Interface is initialized (see "Bootup Sequence", p. 3-3).

Since MIB attributes should usually not be modified, the main reason for accessing the MIB is to acquire data for monitoring, controlling, and statistics-gathering purposes.

[9] As specified in FDDI Station Management, rev. 6.2.

Reading the Value of a MIB Attribute

To determine the current value of a MIB attribute:

1. issue the command Get MIB Attribute (an RC request described on 4-23)
2. read the returned MIB Attribute Response (an RC response described on p. 4-56)

Setting the Value of a MIB Attribute

To assign a value to a MIB attribute, issue the command Set MIB Attribute (p. 4-26). Since this is an RC directive, no response is returned by the 4211. The command cannot be used to change "read-only" attributes.

Changing the MIB attributes is not a trivial task. It requires an *expert-level* understanding of revision 6.2 of the FDDI Station Management specifications. Therefore, it is not recommended for most applications.

Resetting the 4211 or cycling the power restores the default MIB attributes.

SMT FRAME AND EVENT TRACING

If the host needs to monitor the network closely, it can instruct the 4211 to upload SMT frames and/or events as they are received by the board. See the SMT Trace directive (p. 4-35) and SMT Event Indication (p. 4-58) for details.

Like the commands for accessing the MIB, the SMT trace commands require a good understanding of SMT rev. 6.2 specifications. In particular, the 6.2 SMT manual defines the attributes associated with SMT frames and events.

This page is intentionally left blank.

CHAPTER 4
FDDI COMMANDS

OVERVIEW

This chapter describes the 4211-specific command set, which includes transmit, receive, and other network-related commands. In addition, the 4211 supports all commands in the generic RC command set, as documented in the *Common Boot and RC Interface User's Guide*.

As discussed in the RC manual, host-to-board directed information is described by **requests** and **directives**. Requests have responses associated with them (a reply in the board-to-host direction). Directives, on the other hand, have no associated reply.

Board-to-host directed information is described by **responses** and **Indications**. Responses are paired with requests that are host-initiated. Indications are asynchronous events which have no pre-initiated host-to-board trigger.

A summary of the 4211's controller-specific command set is shown in Table 4-1.

Table 4–1 . 4211 FDDI Command Set

REQUESTS AND DIRECTIVES			
Command ID	Name	Type	Page # †
0x101	Transmit Raw Data	Request	4-6
0x102	Transmit Datagram	Request	4-10
0x103	Set Up Receive Buffers	Directive	4-14
0x104	Request Station Address	Request	4-17
0x105	Set Station Address	Directive	4-18
0x106	Ring Control	Directive	4-20
0x107	Perform Maintenance	Directive	4-21
0x108	Get MIB Attribute	Request	4-23
0x109	Set MIB Attribute	Directive	4-26
0x10A	Set CAM Address	Request	4-31
0x10B	Clear CAM Address	Request	4-33
0x10C	Flush CAM Addresses	Directive	4-34
0x10D	SMT Trace	Directive	4-35
0x10E	Set Hardware Parameters	Directive	4-37
0x10F	Get Hardware Parameters	Request	4-41
0x110	Get CAM Address	Request	4-43
RESPONSES AND INDICATIONS			
0x181	Tx Response	Response	4-45
0x182	Rx Frame	Indication	4-47
0x183	(reserved)	–	–
0x184	Report Station Address	Response	4-50
0x185	Ring Status Indication	Indication	4-52
0x186	(reserved)	–	–
0x187	CAM Address Response	Response	4-54
0x188	MIB Attribute Response	Response	4-56
0x189	SMT Event Indication	Indication	4-58
0x18A	Report Hardware Parameters	Response	4-59
0x18B	Report CAM Address	Response	4-64

† Indicates the page in this manual on which the command description can be found.

COMMONLY USED COMMAND FIELDS

With three exceptions, the fields in a command are explained in the section on that command. The three exceptions are:

- Command Control Block (CCB)
- Channel ID
- Transfer Options Word

Since the above fields occur in many 4211 commands (or all of them, in the case of the CCB), they are explained only once. A description of these fields appears in the next three sections.

NOTE: The following descriptions of the CCB, Channel ID, and Transfer Options Word are identical to those in the *Common Boot and RC Interface User's Guide*. They are repeated in this manual for ease of reference.

Command Control Block Format

The Command Control Block (CCB) is an 8-byte data structure placed at the start of any request, directive, response, or indication sent through RC channels. The data structure of the CCB is shown on p. B-9. Its format is as follows:

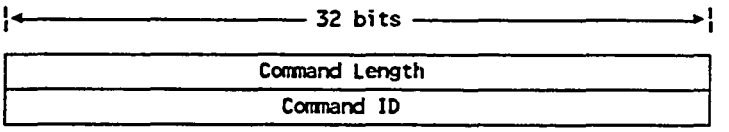


Figure 4-1 . Command Control Block Structure

FIELDS

The fields in the Command Control Block are as follows:

• Command Length (4 bytes)

This field contains the number of bytes in the entire command, including the CCB, the parameters, and any extra padding. This value is used by the reader of a channel to find the next command in the channel.

• Command ID (4 bytes)

This field identifies the specific command (request, directive, response, or indication) to be processed. The command IDs of the 4211-specific command set are shown in the table on p. 4-2.

NOTE: There is no relationship between a "command ID" and a "channel ID".

Channel ID Fields

Many RC commands include a field that identifies a specific channel. The name of the field varies slightly depending on its purpose. For example, the Set Interrupt directive has a "Target Channel ID" field that identifies the channel for which interrupts are being enabled. Likewise, request-type commands (such as Request Statistics) contain the field "ID # of Response Channel" to tell the controller where to write its response.

In all cases, the ID of a channel is the offset (in bytes) at which its channel descriptor starts in shared memory. Examples of channel IDs include:

- The ID of the initial board-to-host channel created at RC boot time. This value is always 0x20, since the channel descriptor of the initial board-to-host channel always begins at an offset of +0x20 (32 bytes) from the start of shared memory.
- The ID of the initial host-to-board channel created at RC boot time. This value is always 0x10, since the channel descriptor of the initial board-to-host channel always begins at an offset of +0x10 (16 bytes) from the start of shared memory.
- The ID of any channel created after RC boot time. The RC Interface assigns an ID to a channel upon successful completion of a Create Channel request (either host-to-board or board-to-host). It returns ID of the new channel via the appropriate response (Report New Host-to-Board Channel or Report New Board-to-Host Channel).

Transfer Options Word

The Transfer Options Word is used to specify what type of VMEbus transaction will be used for data transfers. The bits in the transfer options word indicate whether the transfer should be 16 or 32 bits wide and which address modifier should be used.

Note that the Transfer Options Word is defined as a 32-bit unsigned integer, even though only the least significant word of the field (bits 0 -15) is used to specify options.[10] It is recommended that the host treat the most significant word in the field (bits 16 - 31) as a reserved 16-bit field.

The format of the Transfer Options Word is shown in Figure 4-2.

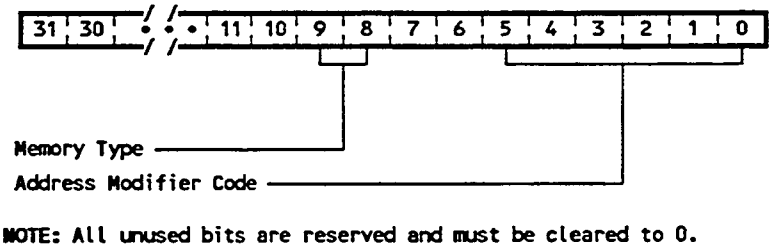


Figure 4–2 . Transfer Options Word for Data Transfers

[10] This is due to the way that static RAM is configured on the 4211.

Bits 0–5 Address Modifier:

This field contains the VMEbus address modifier used by the controller for all VMEbus data transfers associated with this data transfer. It is not modified in any way by the controller. Valid entries in this field are listed in the following table:

Table 4–2 . Address Modifiers Allowed in Data Transfers

CODE	FUNCTION
0x09	Ext. Non-Privileged Data Access
0x0B	Ext. Non-Privileged Block Transfer
0x0D	Ext. Supervisory Data Access
0x0F	Ext. Supervisory Block Transfer
0x29	Short Non-Privileged I/O
0x2D	Short Supervisory I/O
0x39	Std. Non-Privileged Data Access
0x3B	Std. Non-Privileged Block Transfer
0x3D	Std. Supervisory Data Access
0x3F	Std. Supervisory Block Transfer

Bits 6–7 Reserved:

These bits are reserved and must be cleared to 0.

Bits 8–9 Memory Type:

This two-bit field specifies the width of the data transfer. Permitted values are as follows:

Table 4–3 . Memory Type for Data Transfers

Bit 9	Bit 8	MEMORY TYPE
0	0	(reserved)
0	1	16-bit transfers
1	0	32-bit transfers
1	1	(reserved)

Bits 10–31 Reserved:

These bits are reserved and must be cleared to 0.

EXAMPLE:

For a DMA transfer in a SUN 3/260 4.1 operating system, the following transfer options word would be appropriate:

Bit #	Bits 31 - 10	9	8	7	6	5	4	3	2	1	0
Setting	(all 0's)	1	0	0	0	1	1	1	1	0	1

This specifies 32-bit addressing and a VME address modifier of 0x3D.

TRANSMIT RAW DATA

Group: FDDI
Type: Request
Command ID: 0x101

TRANSMIT RAW DATA
Group: FDDI
Type: Request
Command ID: 0x101

Transmit Raw Data is a non-FDDI-standard request. It provides a mechanism to pass one frame from host-to-fiber with absolutely no interpretation or analysis performed by the board. This ability is necessary, for example, in applications where the Logical Link Control (LLC) implementation is responsible for all FDDI frame construction and data encapsulation.

Data for the frame can be "gathered" from discontiguous blocks of host memory. In this case, the host typically writes the frame header in the first fragment and the actual data in subsequent fragments. For more information, see "Scatter/Gather Capabilities", 3-12.

If necessary, the host may pad the frame header to align it on a word boundary. For example, the MAC and LLC frame headers defined on p. B-8 include a 3-byte pad to longword-align the headers. Use of padding will depend on your system requirements. If used, however, it should not be included in the "Total # of Bytes for Transmission" and "Physical Address of Data Fragment" fields.

After executing Transmit Raw Data, the 4211 issues a Tx Response (p. 4-45) if it has been enabled to do so in the Request Class field.

The data structure of the Transmit Raw Data request is shown on p. B-13. Its format is as follows:

SIZE (in bytes)	DESCRIPTION
8	Command Control Block
4	ID # of Response Channel
4	Command Tag
4	Request Class
4	Transfer Options Word
4	Total number of bytes for transmission
4	Number of fragments for transmission
4	Physical Address of data fragment 1
4	Length in bytes of data fragment 1
⋮	⋮
4	Physical Address of data fragment n
4	Length in bytes of data fragment n

Figure 4–3 . Transmit Raw Data Request

TRANSMIT RAW DATA

Group: FDDI
Type: Request
Command ID: 0x101

TRANSMIT RAW DATA
Group: FDDI
Type: Request
Command ID: 0x101

FIELDS

The fields in the Transmit Raw Data request are as follows:

• Command Control Block (8 bytes)

This field contains the Command Control Block, as defined on p. 4-3. The command ID must be set to 0x101 for the Transmit Raw Data request.

• ID # of Response Channel (4 bytes)

If a response is enabled for this command (see Request Class field below), this field identifies the board-to-host channel to which the 4211 will write the Tx Response. If this field is 0, the 4211 will use the default response channel (i.e. the initial board-to-host channel created at RC boot time).

• Command Tag (4 bytes)

This field contains a host-supplied tag generated by the device driver. It is usually the address of a host memory descriptor or the virtual address of a data buffer. This value is useful for releasing host memory resources associated with the transmission. The 4211 echoes the command tag back to the host via the Tx Response (see p. 4-45), provided that the response bit is set in the Request Class field.

• Request Class (4 bytes)

This field determines the type of operation or response required by the transmit request.

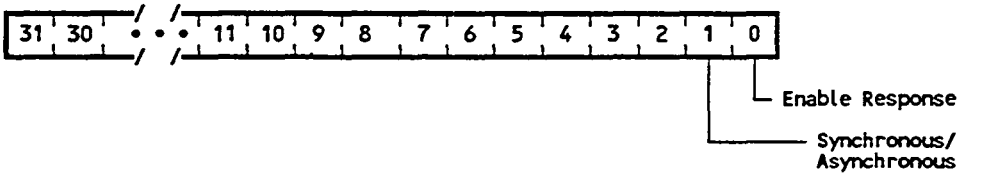


Figure 4–4 . Request Class Field for Transmit Raw Data

Bit 0 Enable Response:

Setting this bit to 1 causes the 4211 to issue a Tx Response after executing the command. Clearing the bit disables the response.

TRANSMIT RAW DATA

Group: FDDI
Type: Request
Command ID: 0x101

TRANSMIT RAW DATA
Group: FDDI
Type: Request
Command ID: 0x101

Bit 1 Synchronous/Asynchronous:

This bit is defined as follows:

0 = send data out as a synchronous transmission
1 = " " " " " asynchronous "

Note that the controller does not force the Frame Control field to conform to this bit setting. In other words, it is possible for the host to send an asynchronous frame in a synchronous frame time. However, this is not in accordance with FDDI standards and is therefore **not** a recommended practice.

For more information synchronous vs. asynchronous transmissions, see the discussion on p. 3-7.

Bits 2 – 31 Reserved:

These bits are reserved and must be cleared to 0 by the host.

• Transfer Options Word (4 bytes)

This field contains the transfer options word, as defined on p. 4-4. The data transfer options in this field apply to all data fragments associated with this command.

• Total # of Bytes for Transmission (4 bytes)

This value is the total number of bytes for transmission, including the frame header.

NOTE: If the frame header is preceded by padding to word-align the header, the pad should ***not*** be included in the byte count.

• Number of Fragments for Transmission (4 bytes)

This value specifies the number of fragments (i.e. Physical Address/Length pairs) following this field.

• Physical Address of Data Fragment *n* (4 bytes)

This field contains the memory address which the 4211's DMA channel uses to locate and move Data Fragment *n* onboard.

NOTE: If the frame header is preceded by padding to word-align the header, the physical address of Data Fragment 1 must point ***past*** the pad to the Frame Control field.

TRANSMIT RAW DATA

Group: FDDI
Type: Request
Command ID: 0x101

TRANSMIT RAW DATA

Group: FDDI
Type: Request
Command ID: 0x101

• Length in Bytes of Data Fragment *n* (4 bytes)

This field contains the number of bytes in Data Fragment *n*.

TRANSMIT DATAGRAM

Group: FDDI
Type: Request
Command ID: 0x102

TRANSMIT DATAGRAM

Group: FDDI
Type: Request
Command ID: 0x102

The Transmit Datagram request is used to pass one IP datagram over FDDI in automatic conformance with RFC 1042 and RFC 1103. The FDDI headers are built by the 4211's firmware and should not be supplied by the host application. The headers built by the firmware consist of the following fields (in order of occurrence):

- 13-byte MAC header (FC field, destination address, and source address)
- 3-byte LLC header (DSAP, SSAP, and Control). The contents of these bytes are as follows:
  - DSAP = 170 (0xAA)
  - SSAP = 170 (0xAA)
  - Control = 3 (0x03)
- 3-byte SNAP header contains all 0's
- 2-byte Ethertype

Data for the frame can be "gathered" from discontiguous blocks of host memory. For more information on this feature, see "Scatter/Gather Capabilities", 3-12.

After executing Transmit Datagram, the 4211 issues a Tx Response (p. 4-45) if it has been enabled to do so in the Request Class field.

The data structure of the Transmit Datagram request is shown on p. B-13. Its format is as follows:

SIZE (in bytes)	DESCRIPTION
8	Command Control Block
4	ID # of Response Channel
4	Command Tag
6	Destination Address
2	Ethertype
4	Request Class
4	Transfer Options Word
4	Total number of bytes for transmission
4	Number of fragments for transmission
4	Physical Address of data fragment 1
4	Length in bytes of data fragment 1
⋮	⋮
4	Physical Address of data fragment n
4	Length in bytes of data fragment n

Figure 4–5 . Transmit Datagram Request

TRANSMIT DATAGRAM

Group: FDDI
Type: Request
Command ID: 0x102

TRANSMIT DATAGRAM
Group: FDDI
Type: Request
Command ID: 0x102

FIELDS

The fields in the Transmit Datagram request are as follows:

• Command Control Block (8 bytes)

This field contains the Command Control Block, as defined on p. 4-3. The command ID must be set to 0x102 for the Transmit Datagram request.

• ID # of Response Channel (4 bytes)

If a response is enabled for this command (see Request Class field below), this field indicates which board-to-host channel the controller should use when it responds to the command. If the field is 0, then the controller will use the default response channel. The default response channel is the initial board-to-host channel created at RC boot time.

• Command Tag (4 bytes)

This field contains a host-supplied tag generated by the device driver. It is usually the address of a host memory descriptor or the virtual address of a data buffer. This value is useful for releasing host memory resources associated with the transmission. The 4211 echoes the command tag back to the host via a Tx Response (see p. 4-45) if the response bit is set in the Request Class field.

• Destination Address (6 bytes)

This field identifies the destination address for insertion into the FDDI MAC header constructed by the firmware. The bit order of the destination address must be "big-endian", meaning that it is in Internet (or "canonical" order), with the Group bit positioned as the low-order bit of the first octet.

FDDI addresses as they appear on the fiber (which is in IEEE or "little-endian" order) have the Group bit in the high order, or most-significant, bit position.

NOTE: The Transmit Datagram command assumes that the source address is already stored in onboard firmware. The source address is either the factory default address or a user-specified address. The factory default address is obtained by issuing Request Station Address (p. 4-17). To specify a different station address, execute the Set Station Address directive (p. 4-18).

• Ethertype (2 bytes)

This field contains the 2-byte Ethertype field that will be inserted into the Logical Link Control SNAP header (RFC 1103) constructed by the 4211.

• Request Class (4 bytes)

TRANSMIT DATAGRAM
Group: FDDI
Type: Request
Command ID: 0x102

TRANSMIT DATAGRAM
Group: FDDI
Type: Request
Command ID: 0x102

This field determines the type of operation or response required by the transmit request.

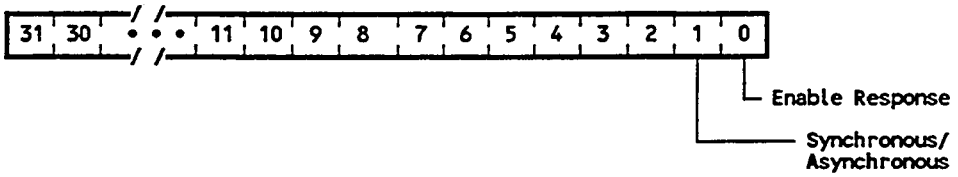


Figure 4–6 . Request Class Field for Transmit Datagram

Bit 0 Enable Response:

Setting this bit to 1 causes the 4211 to issue a Tx Response after executing the command. Clearing the bit disables the response.

Bit 1 Synchronous/Asynchronous:

This bit is defined as follows:

0 = send data out as a synchronous transmission
1 = " " " " " asynchronous "

Note that the controller does not force the Frame Control field to conform to this bit setting. In other words, it is possible for the host to send an asynchronous frame in a synchronous frame time. However, this is not in accordance with FDDI standards and is therefore **not** a recommended practice.

For more information synchronous vs. asynchronous transmissions, see the discussion on p. 3-7.

Bits 2 – 31 Reserved:

These bits are reserved and must be cleared to 0 by the host.

• Transfer Options Word (4 bytes)

This field contains the transfer options word, as defined on p. 4-4. The data transfer options in this field apply to all data fragments associated with this command.

• Total # of Bytes for Transmission (4 bytes)

This value is the total number of bytes for transmission, excluding the frame header (i.e. data only).

TRANSMIT DATAGRAM

Group: FDDI
Type: Request
Command ID: 0x102

TRANSMIT DATAGRAM
Group: FDDI
Type: Request
Command ID: 0x102

- **Number of Fragments for Transmission (4 bytes)**

  This value specifies the number of fragments (i.e. Physical Address/Length pairs) following this field.

- **Physical Address of Data Fragment *n* (4 bytes)**

  This field contains the memory address which the 4211's DMA channel uses to locate and move Data Fragment *n* onboard.

- **Length in Bytes of Data Fragment *n* (4 bytes)**

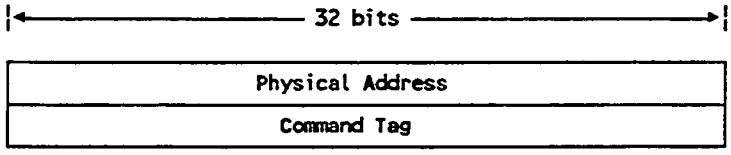
  This field contains the number of bytes in Data Fragment *n*.

SET UP RECEIVE BUFFERS

Group: FDDI
Type: Directive
Command ID: 0x103

SET UP RECEIVE BUFFERS
Group: FDDI
Type: Directive
Command ID: 0x103

The Set Up Receive Buffers directive supplies the 4211 with a buffer list of host memory resources for received data. The buffer list is a block of contiguous memory containing "buffer list elements". The data structure of a buffer list element is shown on p. B-13. Its format is as follows:



Buffer List Element

The Physical Address of a buffer list element is used by the 4211 to DMA data to the buffer. The address must be aligned on a longword boundary. The Command Tag of a buffer list element is a host-assigned value which will be echoed back by the controller via a Rx Frame indication (p. 4-47).

The host delimits the end of the buffer list by writing the special value 0xFFFFFFFF into the Physical Address field of the element beyond the last valid buffer descriptor. For additional information on the buffer list, see pages 3-8 through 3-12.

The 4211 does not pass a response message to the host for the Set Up Receive Buffers directive.

The data structure of the directive is shown on p. B-13. Its format is as follows:

SIZE (in bytes)	DESCRIPTION
8	Command Control Block
4	Transfer Options Word for Buffer List
4	Physical Address of Buffer List
4	Length in Bytes of Buffer List
4	Rx Load Adjustment
4	Transfer Options Word (for all buffers)
4	Length in Bytes of Each Buffer

Figure 4–7 . Set Up Receive Buffers Directive

SET UP RECEIVE BUFFERS

Group: FDDI
Type: Directive
Command ID: 0x103

SET UP RECEIVE BUFFERS
Group: FDDI
Type: Directive
Command ID: 0x103

FIELDS

The fields in the Set Up Receive Buffers directive are as follows:

• Command Control Block (8 bytes)

This field contains the Command Control Block, as defined on p. 4-3. The command ID must be set to 0x103 for the Set Up Receive Buffers directive.

• Transfer Options Word for Buffer List (4 bytes)

This field contains the transfer options word, as defined on p. 4-4. The 4211 uses these transfer options to DMA the buffer list onboard. It is not used to transfer individual buffer list elements.

• Physical Address of Buffer List (4 bytes)

This is the address that the 4211 uses to DMA the buffer list onboard. It must be aligned on a longword boundary.

• Length in Bytes of Buffer List (4 bytes)

This field contains the number of bytes for the 4211 to DMA onboard. Each buffer list element is 8 bytes long. Therefore, to allow for the delimiter, the number of bytes in the buffer list is:

$$8 * (N + 1)$$

(where N is the number of receive buffers in the list)

The recommended maximum list size is 2 Kbytes. Creating a larger list will result in excessive overhead for the 4211.

• Rx Load Adjustment (4 bytes)

This value specifies the maximum number of received frames that the 4211 will process simultaneously. It enables the 4211 to update its onboard copy of the buffer list more often during periods of heavy network traffic.

For example, placing 0x10 in this field causes the 4211 to process no more than 16 incoming frames at once. Also, in the event that it has to process 16 frames simultaneously, the 4211 will refresh its copy of the buffer list after it finishes processing the frames. For a detailed discussion of this feature, see "Balancing the Load of Rx Frame Processing", p. 3-11.

The set of valid adjustment values = {0x1 .. 0x40}. This is because the 4211 can currently process up to 64 (0x40) received frames at once. Entering a value between 0x41 — 0xFF (inclusive) causes the 4211 to use an adjustment value of 0x40. Clearing the field to 0 or entering any value greater

SET UP RECEIVE BUFFERS

Group: FDDI
Type: Directive
Command ID: 0x103

SET UP RECEIVE BUFFERS
Group: FDDI
Type: Directive
Command ID: 0x103

than 0xFF disables the adjustment feature.

• Transfer Options Word for All Buffers (4 bytes)

This field contains the transfer options word, as defined on p. 4-4. The 4211 uses these transfer options to transfer data to the individual receive buffers in the buffer list.

• Length in Bytes of Each Buffer (4 bytes)

This field specifies the size, in bytes, of the individual receive buffers in the list. All buffers in the list must be of this length. The buffer size should be between 512 bytes and 8K bytes (inclusive) and must be a longword multiple (i.e. divisible by 4).

The recommended minimum buffer size is 1 Kbyte. Using smaller buffers will result in excessive bandwidth to transfer the frames across the bus.

REQUEST STATION ADDRESS

Group: FDDI
Type: Request
Command ID: 0x104

REQUEST STATION ADDRESS
Group: FDDI
Type: Request
Command ID: 0x104

Request Station Address is used to obtain the controller's current station address. Interphase assigns a unique address to each individual 4211. The host may change this factory default address using the Set Station Address directive (p. 4-18).

The controller returns its current address by issuing Report Station Address (p. 4-50).

The data structure of Request Station Address is shown on p. B-14. Its format is as follows:

SIZE (in bytes)	DESCRIPTION
8	Command Control Block
4	ID # of Response Channel
4	Command Tag

Figure 4–8 . Request Station Address

FIELDS

The fields in the Request Station Address command are as follows:

• Command Control Block (8 bytes)

This field contains the Command Control Block, as defined on p. 4-3. The command ID must be set to 0x104 for Request Station Address.

• ID # of Response Channel (4 bytes)

This field identifies the board-to-host channel to which the 4211 should write the Report Station Address response. If the field is 0, then the controller will use the default response channel. The default response channel is the initial board-to-host channel created at RC boot time.

• Command Tag (4 bytes)

This field contains a host-supplied tag that will be echoed by the controller in any response to the command.

SET STATION ADDRESS

Group: FDDI
Type: Directive
Command ID: 0x105

SET STATION ADDRESS
Group: FDDI
Type: Directive
Command ID: 0x105

The Set Station Address directive is used to change the 4211's default station address stored in non-volatile RAM (NOVRAM). For your protection, the NOVRAM also contains a write-protected copy of the factory-assigned station address. The Set Station Address directive can be used to restore the factory-assigned address from this backup copy, if necessary.

After the 4211 executes the directive, the host must reset the board or cycle the power in order to make the new address effective.

The controller does not pass a response message to the host for the Set Station Address directive. To verify the updated station address, issue a Request Station Address to the 4211 (p. 4-17), and then read the controller-generated Report Station Address response (p. 4-50).

The data structure of the directive is shown on p. B-14. Its format is as follows:

SIZE (in bytes)	DESCRIPTION
8	Command Control Block
6	Station Address
2	Type

Figure 4–9 . Set Station Address Directive

FIELDS

The fields in the Set Station Address directive are as follows:

• Command Control Block (8 bytes)

This field contains the Command Control Block, as defined on p. 4-3. The command ID must be set to 0x105 for the Set Station Address directive.

• Station Address (6 bytes)

This field sets the desired station address. The address must be represented using the format specified by Internet Protocol (IP). That is, it should be stored byte-wise, with the Group bit positioned as the low order bit of the first octet.

For the sake of efficient data transmission, the address given with this command is bit-reversed within each byte before being written to onboard NOVRAM. Thus, the effective MAC address for the station (i.e., the one placed into the MAC header by a Transmit Datagram request) is drawn directly out of NOVRAM without requiring modification for FDDI format.

The Short Addressing mode, which uses a 16-bit address instead of the more common 48-bit long

SET STATION ADDRESS

Group: FDDI
Type: Directive
Command ID: 0x105

SET STATION ADDRESS
Group: FDDI
Type: Directive
Command ID: 0x105

addresses, is not currently supported.

- **Type (2 bytes)**

This field specifies how the onboard address NOVRAM is to be manipulated. Legal values are as follows:

Table 4–4 . Type Codes in Set Station Address Directive

CODE	PURPOSE
0x0	Restore the factory default station address. This option causes any value in the Station Address field to be ignored.
0x1	Change the current station address to the value specified in the Station Address field.

RING CONTROL

Group: FDDI
Type: Directive
Command ID: 0x106

RING CONTROL
Group: FDDI
Type: Directive
Command ID: 0x106

The Ring Control directive is used to connect or disconnect the 4211 from the FDDI ring. The 4211 does not insert itself into an FDDI network unless it is explicitly instructed to do so.

Issuing this directive causes the board to automatically perform all Station Management functions necessary for entering an FDDI network in accordance with the X3T9.5 standards.

The 4211 does not pass a response message to the host for the Ring Control directive. However, it issues a Ring Status Indication asynchronously whenever the FDDI ring changes state. Thus, if the host succeeds in its attempt to connect to (or disconnect from) the ring, the 4211 will report the new ring status by issuing a Ring Status Indication. If the attempt fails (or the 4211 is already in the state specified by the Ring Control directive), the indication is *not* issued. For more information, see "Connecting the Station to the Ring", p. 3-6, and the description of Ring Status Indication on p. 4-52.

The data structure of the Ring Control directive is shown on p. B-14. Its format is as follows:

SIZE (in bytes)	DESCRIPTION
8	Command Control Block
4	Ring Control Options

Figure 4–10 . Ring Control Directive

FIELDS

The fields in the Ring Control directive are as follows:

• Command Control Block (8 bytes)

This field contains the Command Control Block, as defined on p. 4-3. The command ID must be set to 0x106 for the Ring Control directive.

• Ring Control Options (4 bytes)

Two legal values are currently defined for this field:

0x1 Connect to the ring
0x2 Disconnect from the ring

PERFORM MAINTENANCE

Group: FDDI
Type: Directive
Command ID: 0x107

PERFORM MAINTENANCE
Group: FDDI
Type: Directive
Command ID: 0x107

It is convenient to be capable of generating specific FDDI line states upon demand in order to debug certain classes of network problems. The Perform Maintenance directive places the 4211 in a maintenance state in which generate a desired line state. The command can be reissued to remove the controller from the maintenance state.

The controller does not pass a response message to the host for the Perform Maintenance directive.

The data structure of the directive is shown on p. B-14. Its format is as follows:

SIZE (in bytes)	DESCRIPTION
8	Command Control Block
4	Maintenance Control Options
4	Maintenance State

Figure 4–11 . Perform Maintenance Directive

FIELDS

The fields in the Perform Maintenance directive are as follows:

• Command Control Block (8 bytes)

This field contains the Command Control Block, as defined on p. 4-3. The command ID must be set to 0x107 for the Perform Maintenance directive.

• Maintenance Control Options (4 bytes)

Two values are currently defined for this field:

0x0 Put the PCM state machine into the Maintenance State and issue a specified line state.

0x1 Remove the PCM state machine from the Maintenance State and return to the OFF state.

PERFORM MAINTENANCE
Group: FDDI
Type: Directive
Command ID: 0x107

PERFORM MAINTENANCE
Group: FDDI
Type: Directive
Command ID: 0x107

• Maintenance State (4 bytes)

This field is used to specify a line state when the Maintenance Control field value is 0x0. Legal entries in this field are as follows:

Table 4–5 . Line States Available for Perform Maintenance Directive

VALUE	STATE
0x4	Master
0x8	Idle
0x10	Halt
0x20	Quiet
0x100	Transmit PHY Data Request

GET MIB ATTRIBUTE

Group: FDDI
Type: Request
Command ID: 0x108

GET MIB ATTRIBUTE
Group: FDDI
Type: Request
Command ID: 0x108

Get MIB Attribute is a multi-purpose request which can be used to obtain an element (or group of elements) stored in the Management Information Base (MIB) onboard the controller. Each MIB attribute or group has a specific name.

In order to use this command, you must be familiar with the TLV parameter structure of MIB attributes. For details, refer to Section 6 ("Services") of the rev. 6.2 SMT specifications.

The 4211 returns the requested information to the host by issuing a MIB Attribute Response (p. 4-56).

The data structure of Get MIB Attribute is shown on p. B-14. Its format is as follows:

SIZE (in bytes)	DESCRIPTION
8	Command Control Block
4	ID # of Response Channel
4	Command Tag
4	Attribute Name
4	Data Area Needed for Attribute(s)
4	Attribute Index

Figure 4–12 . Get MIB Attribute Request

FIELDS

The fields in the Get MIB Attribute request are as follows:

• Command Control Block (8 bytes)

This field contains the Command Control Block, as defined on p. 4-3. The command ID must be set to 0x108 for the Get MIB Attribute request.

• ID # of Response Channel (4 bytes)

This field identifies the board-to-host channel to which the 4211 should write the MIB Attribute Response. If the field is 0, then the controller will use the default response channel. The default response channel is the initial board-to-host channel created at RC boot time.

GET MIB ATTRIBUTE
Group: FDDI
Type: Request
Command ID: 0x108

GET MIB ATTRIBUTE
Group: FDDI
Type: Request
Command ID: 0x108

• Command Tag (4 bytes)

This field may contain a host-generated command tag, if needed for the application. The 4211 does not use the tag in any way. It is simply echoed back to the host in the MIB Attribute Response.

• Attribute Name (4 bytes)

This field contains the Parameter_Type of the attribute or MIB group, as defined in SMT rev. 6.2 specifications. The high byte contains the object and sub-object ID, and the lower byte is the attribute registration ID. For example, to request the attribute fddiSMTStationid, use 0x100B in this field.

• Data Area Needed for Attribute(s) (4 bytes)

This field tells the controller how many bytes of on-board data area it needs to handle the requested attribute(s). The host should allow 36 (0x24) bytes for each requested attribute. 36 bytes is the maximum attribute size, as defined in the rev. 6.2 SMT specifications. As noted previously, you must be familiar with the TLV parameter structure of MIB attributes. For details, see Section 6 ("Services") of the rev. 6.2 SMT specifications.

This field should contain the following value:

Data area = 0x24 * the number of attributes being requested

• Attribute Index (4 bytes)

This contains the MAC, Port, Path, or Attachment index as defined in the 6.2 SMT specifications. Valid entries are as follows:

GET MIB ATTRIBUTE

Group: FDDI
Type: Request
Command ID: 0x108

GET MIB ATTRIBUTE
Group: FDDI
Type: Request
Command ID: 0x108

Table 4–6 . Valid Entries in MIB Attribute Index Field

Type of Attribute	Legal Value(s)
SMT	0x0 (SMT attributes do not have an index, so the field must be cleared.)
MAC †	0x1
PATH	0x1 or 0x2
PORT	0x1 or 0x2
ATTACHMENT	0x1 or 0x2

† At the time of this writing [11/19/92], the 4211's firmware supports single–MAC configurations. Dual–MAC operation is supported on the hardware level only.

SET MIB ATTRIBUTE

Group: FDDI
Type: Directive
Command ID: 0x109

SET MIB ATTRIBUTE
Group: FDDI
Type: Directive
Command ID: 0x109

The Set MIB Attribute directive allows the host to assign a value to a specific attribute in the 4211's on-board Management Information Base (MIB). The command can only be used to change attributes with read/write access, as defined in revision 6.2 of the FDDI Station Management specifications.

Since the default settings of the 4211's MIB attributes are correct for most applications, this command is *not* required in most drivers. See "Accessing the MIB", p. 3-14, for more information.

CAUTION

Changing the MIB attributes is not a trivial task. It requires an EXPERT-LEVEL understanding of revision 6.2 of the FDDI Station Management specifications.

In order to use this command, you must be familiar with the TLV parameter structure of MIB attributes. For details, refer to Section 6 ("Services") of the rev. 6.2 SMT specifications.

If SMT rejects the proposed MIB attribute value, it issues a Request Denied Frame (RDF) or Parameter Management Frame (PMF). In this event, the MIB attribute is not changed. Instead, the 4211 informs the host about the RDF or PMF by issuing Report Printf Call (an indication in the generic RC command set). See "Error Messages" (p. 5-4) for details. Note that the absence of a returned RDF or PMF does *not* guarantee that the new value of the MIB attribute is valid.

MIB attributes are stored in static RAM on the 4211. Cycling the power causes the 4211 to restore the default MIB attributes.

The controller does not pass a response message to the host for the Set MIB Attribute directive. If you wish to check the new attribute definition, issue a Get MIB Attribute request.

The data structure of Set MIB Attribute is shown on p. B-14. Its format is shown in Figure 4-13.

SET MIB ATTRIBUTE

Group: FDDI
Type: Directive
Command ID: 0x109

SET MIB ATTRIBUTE
Group: FDDI
Type: Directive
Command ID: 0x109

SIZE (in bytes)	DESCRIPTION
8	Command Control Block
4	Attribute Name
4	Data Area Needed for Attribute
4	Attribute Index
n	Attribute Definition (Variable-length)

Figure 4–13 . Set MIB Attribute Directive

FIELDS

The fields in the Set MIB Attribute directive are as follows.

• Command Control Block (8 bytes)

This field contains the Command Control Block, as defined on p. 4-3. The command ID must be set to 0x109 for the Set MIB Attribute directive.

• Attribute Name (4 bytes)

This field contains the Parameter_Type of the attribute, as defined in SMT rev. 6.2 specifications. The high byte contains the object and sub-object ID, and the lower byte is the attribute registration ID. For example, to specify the attribute fddiSMTConnectionPolicy, write 0x101B to this field.

• Data Area Needed for Attribute (4 bytes)

This field should contain the attribute's Parameter_Length, as defined in the 6.2 SMT specs. For example, if you are setting the attribute fddiSMTConnectionPolicy, write 0x0004 to this field.

• Attribute Index (4 bytes)

This field contains the MAC, Port, Path, or Attachment index as defined in the 6.2 SMT specifications. Valid entries are shown in Table 4-7.

SET MIB ATTRIBUTE

Group: FDDI
Type: Directive
Command ID: 0x109

SET MIB ATTRIBUTE
Group: FDDI
Type: Directive
Command ID: 0x109

Table 4-7 . Valid Entries in MIB Attribute Index Field

Type of Attribute	Legal Value(s)
SMT	0x0 (SMT attributes do not have an index, so the field must be cleared.)
MAC †	0x1
PATH	0x1 or 0x2
PORT	0x1 or 0x2
ATTACHMENT	0x1 or 0x2

† At the time of this writing [11/19/92], the 4211's firmware supports single-MAC configurations. Dual-MAC operation is supported on the hardware level only.

• Attribute Definition (Variable-Length)

This field contains the new attribute definition. The host must write the data as a byte stream, using exactly the same TLV encoding as given in the 6.2 SMT specifications. No termination is necessary.

SET MIB ATTRIBUTE

Group: FDDI
Type: Directive
Command ID: 0x109

SET MIB ATTRIBUTE
Group: FDDI
Type: Directive
Command ID: 0x109

EXAMPLE:

As an example, suppose you wish to set the attribute **fddiSMTConnectionPolicy**. The TLV encoding of this attribute (as defined on pages 47 - 48 of the SMT rev. 6.2 specifications) is shown below.

Parameter_Type	X'10 1B'
Parameter_Length	X'00 04'
Pad	X'00 00'
Parameter_Value	X'00 00'-X'00 7F'

To assign a new value to fddiSMTConnectionPolicy, execute a Set MIB Attribute command with the fields filled in as follows:

Table 4–8 . Example Set MIB Command (fddiSMTConnectionPolicy)

Field	Longword #	Contents	Comments
Command Control Block (8 bytes)	0x0 0x1	0x00000038 0x00000109	Total command length is 0x38 bytes. Command ID is 0x109.
Attribute Name (4 bytes)	0x2	0x0000101B	This is the attribute's Parameter_Type, as defined in the 6.2 SMT specs.
Data Area Needed for Attribute (4 bytes)	0x3	0x00000004	This is the attribute's Parameter_Length, as defined in the 6.2 SMT specs.
Attribute Index (4 bytes)	0x4	0x00000000	SMT attributes have no index (see Table 4-7, p. 4-27).
Attribute Definition (16 bytes for fddiSMTConnectionPolicy)	0x5 0x6 0x7 - 0xD	0x101B0004 0x0000*yyyy* 0x*zzzzzzzz*	Parameter type and length *yyyy* = Attribute value *zzzz...* = Don't care values in TLVParamType

NOTE: If the 4211 undergoes a hard reset, the default parameters will be restored.

To better understand how to fill in the command fields, examine the following C listing. This code produces in host memory a hex dump matching the "Contents" fields shown in the preceding example. It uses the TLVParamStruct structure shown on p. B-21.

SET MIB ATTRIBUTE

Group: FDDI
Type: Directive
Command ID: 0x109

SET MIB ATTRIBUTE
Group: FDDI
Type: Directive
Command ID: 0x109

```
extern      ULONG      *rc_ptr;                    /* host->board channel pointer */
set_smtconnectpolicy(void)
{
ULONG cmd_length,      *cmd_ptr;
USHORT      reject;
FDDI_SET_MIB           *ccb_ptr;
TLVParamType           *smt_ptr;

      reject = get_policy();

      cmd_length =
           (((sizeof(FDDI_SET_MIB))-(sizeof(u32)))+(sizeof(TLVParamType)));
      cmd_lenght = allign_it(cmd_length);       /* insure long word alignment  */
      will_it_fit(cmd_length);                  /* check for link necessary    */
      cmd_ptr = rc_ptr;
      ccb_ptr = (FDDI_SET_MIB *)rc_ptr;
      smt_ptr = (TLVParamType *)&ccb_ptr->attribute;

      ccb_ptr->ccb.len = cmd_length;
      ccb_ptr->ccb.cmd = FDDI_SET_MIB_DIRECTIVE;
      ccb_ptr->attr_name = fddiSMTConnectionPolicy;
      ccb_ptr->attr_length = 0x4;               /* set to 4 per SMT spec       */
      ccb_ptr->attr_index = 0;                  /* index = 0 for smt           */

      smt_ptr->paramType = fddiSMTConnectionPolicy;
      smt_ptr->paramLen = 0x4;                  /* set to 4 per SMT spec       */
      smt_ptr->SMTPARAM16 = reject;

      rc_ptr += (cmd_length/(sizeof(u32)));
      do_h2b( cmd_ptr );                        /* display and execute command */
}
```

The hex code produced by the above code is as follows.

Longword	Data	Explanation
0x0	0x00000038	See field descriptions in Table 4-8.
0x1	0x00000109	
0x2	0x0000101b	
0x3	0x00000004	
0x4	0x00000000	
0x5	0x101b0004	
0x6	0x0000yyyy	
0x7 - 0xD	0xzzzzzzzz	

Diagram 4–1 . C Code and Hex Dump for Set MIB Example

SET CAM ADDRESS

Group: FDDI
Type: Request
Command ID: 0x10A

SET CAM ADDRESS
Group: FDDI
Type: Request
Command ID: 0x10A

Set CAM Address is used to store an address in the 4211's content-addressable memory (CAM). This feature allows the 4211 to receive data addressed to up to 256 different destinations, in addition to the normal base station address embedded in NOVRAM. As frames on the ring move through the 4211, their destination addresses are matched against addresses stored in the CAM. Whenever a match is found, the 4211 copies that frame into onboard memory and passes it up to the host as an ordinary receive indication.

The command will complete with error if the host attempts to set an address which is already present in the CAM or if the CAM already contains 256 addresses. To remove an individual address from the CAM, issue the Clear CAM Address request (p. 4-33). To clear out all CAM addresses, issue the directive Flush CAM Addresses (p. 4-34).

After executing the request, the 4211 issues CAM Address Response (p. 4-54).

The data structure of Set CAM Address is shown on p. B-14. Its format is as follows:

SIZE (in bytes)	DESCRIPTION
8	Command Control Block
4	ID # of Response Channel
4	Command Tag
6	Station Address
2	Reserved [1]

[1] Reserved fields must be cleared to 0 by the host.

Figure 4–14 . Set CAM Address Request

FIELDS

The fields in the Set CAM Address request are as follows:

• Command Control Block (8 bytes)

This field contains the Command Control Block, as defined on p. 4-3. The command ID must be set to 0x10A for the Set CAM Address request.

SET CAM ADDRESS

Group: FDDI
Type: Request
Command ID: 0x10A

SET CAM ADDRESS
Group: FDDI
Type: Request
Command ID: 0x10A

• ID # of Response Channel (4 bytes)

This field identifies the board-to-host channel to which the 4211 should write the CAM Address Response. If the field is 0, then the controller will use the default response channel. The default response channel is the initial board-to-host channel created at RC boot time.

• Command Tag (4 bytes)

This field may contain a host-generated command tag, if needed for the application. The 4211 does not use the tag in any way. It is simply echoed back to the host in the CAM Address Response.

• Station Address (6 bytes)

The station address should be given in Internet bit order. The 4211 will bit-swap it to IEEE bit-transmission order before placing it in the CAM.

CLEAR CAM ADDRESS

Group: FDDI
Type: Request
Command ID: 0x10B

CLEAR CAM ADDRESS
Group: FDDI
Type: Request
Command ID: 0x10B

Clear CAM Address is used to remove a destination address which the host has previously stored in the 4211's content-addressable memory (CAM). After executing the request, the 4211 issues CAM Address Response (p. 4-54). If the specified address is not present in the CAM, the response will flag the error.

The data structure of Clear CAM Address is shown on p. B-14. Its format is as follows:

SIZE (in bytes)	DESCRIPTION
8	Command Control Block
4	ID # of Response Channel
4	Command Tag
6	Station Address
2	Reserved [1]

[1] Reserved fields must be cleared to 0 by the host.

Figure 4–15 . Clear CAM Address Request

FIELDS

The fields in the Clear CAM Address request are as follows:

• Command Control Block (8 bytes)

This field contains the Command Control Block, as defined on p. 4-3. The command ID must be set to 0x10B for the Clear CAM Address request.

• ID # of Response Channel (4 bytes)

This field identifies the board-to-host channel to which the 4211 should write the CAM Address Response. If the field is 0, then the controller will use the default response channel. The default response channel is the initial board-to-host channel created at RC boot time.

• Command Tag (4 bytes)

This field may contain a host-generated command tag, if needed for the application.

• Station Address (6 bytes)

This field contains the station address that is to be removed from the CAM. It should specified in Internet bit order.

FLUSH CAM ADDRESSES

Group: FDDI
Type: Directive
Command ID: 0x10C

FLUSH CAM ADDRESSES

Group: FDDI
Type: Directive
Command ID: 0x10C

The Flush CAM Addresses directive is used to remove *all* addresses that the host stored in the 4211's content-addressable memory (CAM).

The data structure of the directive is shown on p. B-14. Its format is as follows:

SIZE (in bytes)	DESCRIPTION
8	Command Control Block

Figure 4–16 . Flush CAM Addresses Directive

FIELD

The field in the Flush CAM Addresses directive is as follows:

- **Command Control Block (8 bytes)**

  This field contains the Command Control Block, as defined on p. 4-3. The command ID must be set to 0x10C for the Flush CAM Addresses directive.

SMT TRACE

Group: FDDI
Type: Directive
Command ID: 0x10D

SMT TRACE
Group: FDDI
Type: Directive
Command ID: 0x10D

The SMT Trace directive is used to instruct the 4211 to route SMT frames and/or events to the host. The command must be issued twice in order to enable both frame and event tracing. The default is for the 4211 to route neither of these SMT entities to the host.

If the host enables frame tracing, the 4211 will route each incoming SMT frame to the host by issuing an Rx Frame indication (see p. 4-47). If event tracing is enabled, the board routes issues an SMT Event Indication (p. 4-58) to notify the host of each incoming SMT message.

To disable frame or event tracing, the host reissues the directive with the appropriate value in the Trace Control field, as described below.

The data structure of the directive is shown on p. B-14. Its format is as follows:

SIZE (in bytes)	DESCRIPTION
8	Command Control Block
4	Trace Control

Figure 4–17 . SMT Trace Directive

FIELDS

The fields in the SMT Trace directive are as follows:

• Command Control Block (8 bytes)

This field contains the Command Control Block, as defined on p. 4-3. The command ID must be set to 0x10D for the SMT Trace directive.

• Trace Control (4 bytes)

Table 4-9 shows the legal values for this field.

SMT TRACE

Group: FDDI
Type: Directive
Command ID: 0x10D

SMT TRACE
Group: FDDI
Type: Directive
Command ID: 0x10D

Table 4–9 . Values in Trace Control Field

Value	Description
0x1	Turn SMT frame tracing on. Route each incoming SMT message to the host via an Rx Frame indication.
0x2	Turn SMT frame tracing off.
0x3	Turn SMT event tracing on. Route each incoming SMT event to the host via an SMT Event Indication.
0x4	Turn SMT event tracing off.

SET HARDWARE PARAMETERS

Group: FDDI
Type: Directive
Command ID: 0x10E

SET HARDWARE PARAMETERS

Group: FDDI
Type: Directive
Command ID: 0x10E

Set Hardware Parameters allows manipulation of specific bits in various hardware registers on the 4211. These registers control various modes of operation and enable/disable specific functions of the controller. One or any combination of the supported hardware registers may be written in a single directive. No specific ordering of parameters is required.

The controller does not pass a response message to the host for the Set Hardware Parameters directive. The format of the directive is as follows:

SIZE (in bytes)	DESCRIPTION
8	Command Control Block
4	Command Tag
4	Parameter Count
2	Parameter Type
2	Parameter Data
⋮	⋮
2	Parameter Type
2	Parameter Data

Figure 4–18 . Set Hardware Parameters Directive

FIELDS

The fields in the Set Hardware Parameters directive are as follows:

• Command Control Block (8 bytes)

This field contains the Command Control Block, as defined on p. 4-3. The command ID must be set to 0x10E for the Set Hardware Parameters directive.

• Command Tag (4 bytes)

This field contains a host-supplied tag generated by the device driver. It is usually the address of a host memory descriptor or the virtual address of a data buffer.

• Parameter Count (4 bytes)

This field contains the number of parameter changes supplied in the remainder of this command. For each change, there is one pair of Parameter Type and Parameter Data fields.

Example: If there are five pairs of Parameter Type/Parameter Data fields in the directive, the

SET HARDWARE PARAMETERS

Group: FDDI
Type: Directive
Command ID: 0x10E

SET HARDWARE PARAMETERS

Group: FDDI
Type: Directive
Command ID: 0x10E

Parameter Count field should contain 0x5.

• Parameter Type *n* (2 bytes)

The code in this field identifies the type of parameter to be manipulated. Table 4-10 lists the codes for the available registers and functions. As noted previously, no specific ordering of parameters is required.

Table 4–10 . Host-Settable Registers and Functions

CODE	PARAMETER TYPE	VALID PARAMETER DATA
0x0001	CAM Command Register	See Diagram 4-2, p. 4-38
0x0002	Front End Command Register	See Diagram 4-3, p. 4-39
0x0003	(reserved)	------
0x0004	Set Formac Flush Mode	See Table 4-11, p. 4-39
0x0005	Disable/Enable FCS Generation	See Table 4-12, p. 4-39
0x0006	Optical Bypass Relay Control	See Table 4-13, p. 4-40

• Parameter Data *n* (2 bytes)

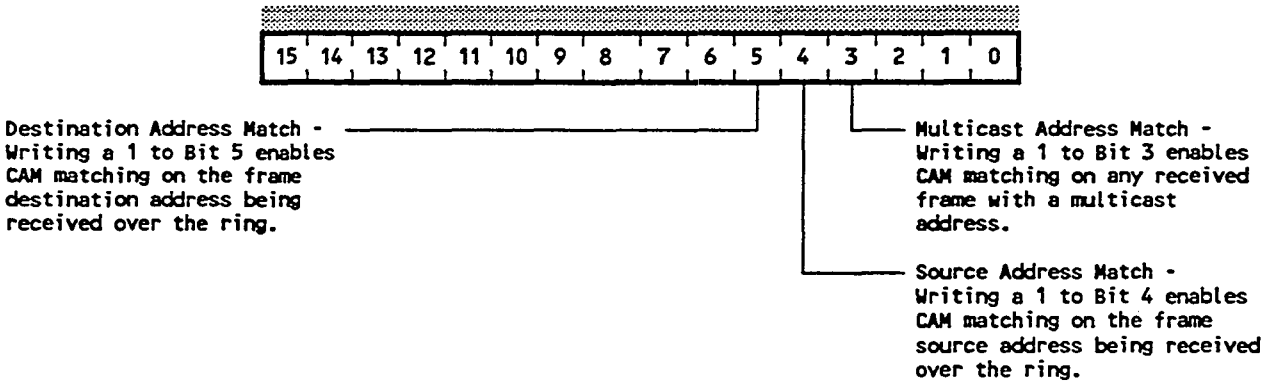
This field contains either: 1) a mapped field of control bits which will be written to a specific register, or 2) a code enabling/disabling a specific function. The data in this field must be valid for the Parameter Type field with which it is paired (i.e. the 2-byte Parameter Type field immediately preceding it). Valid entries for the five parameter types listed in Table 4-10 are described in the following five diagrams and tables.

SET HARDWARE PARAMETERS

Group: FDDI
Type: Directive
Command ID: 0x10E

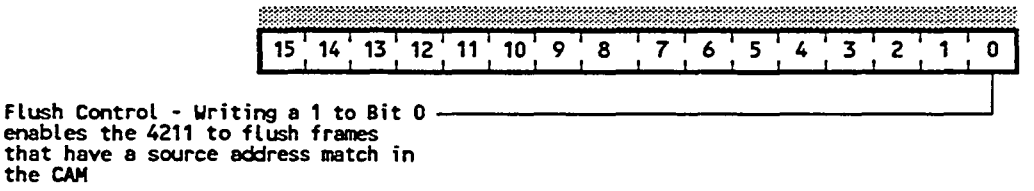
SET HARDWARE PARAMETERS

Group: FDDI
Type: Directive
Command ID: 0x10E



NOTE: Bits which are not referenced above are reserved and must be cleared to 0.

Diagram 4–2 . Host-Settable Bits in CAM Command Register



NOTE: Bits which are not referenced above are reserved and must be cleared to 0.

Diagram 4–3 . Host-Settable Bits in Front End Command Register

Table 4–11 . FORMAC Frame Flush Mode Control

CODE	FORMAC FRAME FLUSH MODE
0x0	No flush if the destination address matches the local MAC address.
0x1	No flush if the destination or source address matches the local MAC address.
0x2	No flush (promiscuous mode)

SET HARDWARE PARAMETERS

Group: FDDI
Type: Directive
Command ID: 0x10E

SET HARDWARE PARAMETERS
Group: FDDI
Type: Directive
Command ID: 0x10E

Table 4–12 . FCS Generation Mode Control

CODE	MODE
0x0	Frame Check Sequence is sent on transmit frames.
0x1	Inhibits the generation of the Frame Check Sequence on transmitted frames.

Table 4–13 . Optical Bypass Relay Control

CODE	MODE
0x0	Enable bypass mode.
0x1	Enable THRU mode.

GET HARDWARE PARAMETERS

Group: FDDI
Type: Request
Command ID: 0x10F

GET HARDWARE PARAMETERS
Group: FDDI
Type: Request
Command ID: 0x10F

Get Hardware Parameters is used to read various hardware registers on the controller. One or any combination of the supported hardware registers may be read in a single request. No specific ordering of parameters is required.

The controller issues Report Hardware Parameters (p. 4-59) in response to this request.

The format of the Get Hardware Parameters is as follows:

SIZE (in bytes)	DESCRIPTION
8	Command Control Block
4	Command Tag
4	Parameter Count
2	Parameter Type
2	Reserved
2	Parameter Type
2	Reserved
⋮	⋮
2	Parameter Type
2	Reserved

[1] Reserved fields must be cleared to 0.

Figure 4-19 . Get Hardware Parameters Request

FIELDS

The fields in the Get Hardware Parameters request are as follows:

• Command Control Block (8 bytes)

This field contains the Command Control Block, as defined on p. 4-3. The command ID must be set to 0x10F for the Get Hardware Parameters request.

• Command Tag (4 bytes)

This field contains a host-supplied tag generated by the device driver. It is usually the address of a host memory descriptor or the virtual address of a data buffer.

GET HARDWARE PARAMETERS

Group: FDDI
Type: Request
Command ID: 0x10F

GET HARDWARE PARAMETERS
Group: FDDI
Type: Request
Command ID: 0x10F

• Parameter Count (4 bytes)

This field contains the number of parameters requested in the remainder of this command. The same parameter can be requested multiple times in the same command, if desired.

• Parameter Type *n* (2 bytes)

The code in this field identifies the type of parameter being requested. Table 4-14 lists the codes for the available registers and functions. As noted previously, no specific ordering of parameters is required.

Table 4–14 . Host-Readable Registers and Functions

CODE	PARAMETER TYPE
0x0001	CAM Command Register
0x0002	Front End Command Register
0x0003	(reserved)
0x0004	Get Formac Flush Mode
0x0005	Disable/Enable FCS Generation
0x0006	Optical Bypass Relay Control

GET CAM ADDRESS

Group: FDDI
Type: Request
Command ID: 0x110

GET CAM ADDRESS
Group: FDDI
Type: Request
Command ID: 0x110

Get CAM Address is used to query a specific entry in the 4211's 256-entry content-addressable memory (CAM). The host returns the requested CAM entry by issuing Report CAM Address (p. 4-64).

The format of Get CAM Address is as follows:

SIZE (in bytes)	DESCRIPTION
8	Command Control Block
4	ID # of Response Channel
4	Command Tag
2	CAM Address
2	Reserved

NOTE: Reserved fields must be cleared to 0 by the host.

Figure 4-20 . Get CAM Address Request

FIELDS

The fields in the Get CAM Address request are as follows:

• Command Control Block (8 bytes)

This field contains the Command Control Block, as defined on p. 4-3. The command ID must be set to 0x110 for the Get CAM Address request.

• ID # of Response Channel (4 bytes)

This field identifies the board-to-host channel to which the 4211 should write the Report CAM Address response. If the field is 0, then the controller will use the default response channel. The default response channel is the initial board-to-host channel created at RC boot time.

• Command Tag (4 bytes)

This field may contain a host-generated command tag, if needed for the application. The 4211 does not use the tag in any way. It is simply echoed back to the host in the CAM Address Response.

GET CAM ADDRESS

Group: FDDI
Type: Request
Command ID: 0x110

GET CAM ADDRESS

Group: FDDI
Type: Request
Command ID: 0x110

• CAM Address (2 bytes)

This field identifies which entry in the 256-entry CAM is being queried. Possible values are 0x0000 — 0x00FF.

Tx RESPONSE

Group: FDDI
Type: Response
Command ID: 0x181

Tx RESPONSE
Group: FDDI
Type: Response
Command ID: 0x181

Tx Response is generated by the 4211 after it executes a Transmit Raw Data or a Transmit Datagram request. It will not be issued unless responses are enabled in the Response Class field of the completed command.

This response indicates whether the 4211 has successfully queued the data for transmission in its one-megabyte communications buffer. For more information, see "Transmitting Data", p. 3-7.

The data structure of Tx Response is shown on p. B-15. Its format is as follows:

SIZE (in bytes)	DESCRIPTION
8	Command Control Block
4	Command Tag
4	Transmit Completion Status

Figure 4-21 . Tx Response

FIELDS

The fields in Tx Response are as follows:

• Command Control Block (8 bytes)

This field contains the Command Control Block, as defined on p. 4-3. The controller sets the command ID field to 0x181 for Tx Response.

• Command Tag (4 bytes)

This field contains the host-supplied command tag from the completed transmit command. It is echoed unchanged back to the host by the controller. Typically, the tag somehow identifies the host memory resources associated with the transmit command.

• Transmit Completion Status (4 bytes)

This field reports whether the 4211 was able to copy the frame specified by the transmit command into its communications buffer. Possible status codes are shown in Table 4-15.

Tx RESPONSE

Group: FDDI
Type: Response
Command ID: 0x181

Tx RESPONSE
Group: FDDI
Type: Response
Command ID: 0x181

Table 4–15 . Transmit Completion Status Field

CODE	MEANING
0x0	Normal. Frame queued for transmission.
0x1	(reserved - not used)
0x2	Ring is down. Frame is discarded.
0x3	VMEbus error. Frame is discarded.

If the status code is 0, the host may reuse any host memory space containing outgoing data associated with the transmit command. Otherwise, the data should be either retained in host memory for later transmission or discarded, depending on the problem.

Rx FRAME

Group: FDDI
Type: Indication
Command ID: 0x182

Rx FRAME

Group: FDDI
Type: Indication
Command ID: 0x182

Rx Frame is used by the 4211 to pass a received frame from fiber-to-host without interpretation or analysis. Any media-specific header data, such as the FDDI MAC header, appears at the beginning of the receive buffer.[11]

In addition to uploading link-level packets, Rx Frame is also used to upload SMT frames, provided the host has enabled SMT frame tracing via the SMT Trace directive (p. 4-35). In this case, the receive buffer will contain an actual SMT packet as it was received from the fiber.

The 4211 writes the Rx Frame indication itself into the default response channel (i.e. the initial board-to-host channel created at RC boot time). On the other hand, the received frame is DMA'ed into a host-resident "receive buffer". See "Receiving Data" (p. 3-8) and the Set Up Receive Buffers directive (p. 4-14) for details. If the frame cannot fit into a single receive buffer, the 4211 "scatters" the frame across multiple discontiguous buffers. For more information, see "Scatter/Gather Capabilities", 3-12.

When the 4211 uploads a received frame to the host, it aligns the frame on a longword boundary. Since the FDDI MAC frame header is 13 bytes long, some padding is necessary to longword-align the frame. Therefore, the 4211 places a 3-byte pad at the beginning of the first frame fragment. Subsequent fragments of that frame are not offset. Their contents are written to the beginning of receive buffers.

The data structure of the Rx Frame indication is shown on p. B-15. Its format is shown in Figure 4-22.

[11] The term "receive buffer" refers to an area of host memory which the host allocates to the 4211 for use when uploading frames to the host. See "Receiving Data" (p. 3-8) and the Set Up Receive Buffers directive (p. 4-14).

Rx FRAME
Group: FDDI
Type: Indication
Command ID: 0x182

Rx FRAME
Group: FDDI
Type: Indication
Command ID: 0x182

SIZE (in bytes)	DESCRIPTION
8	Command Control Block
2	Receive Status
2	Frame Status
2	Length in bytes of received frame
2	Number of fragments in received frame
4	Command tag for data fragment 1
4	Length in bytes of data fragment 1
4	Command Tag for data fragment 2
4	Length in bytes of data fragment 2
⋮	⋮
4	Command Tag for data fragment n
4	Length in bytes of data fragment n

Figure 4–22 . Rx Frame Indication

FIELDS

The fields in the Rx Frame indication are as follows:

• Command Control Block (8 bytes)

This field contains the Command Control Block, as defined on p. 4-3. The controller sets the command ID field to 0x182 for the Rx Frame indication.

• Receive Status (2 bytes)

This field provides information concerning the 4211's ability to receive the frame and move it to the host. Currently, no status codes are defined; therefore, the field is cleared to 0.

• Frame Status (2 bytes)

This field contains the E, A, & C bits as defined in the MAC specification. Its format is shown in Figure 4-23.

Rx FRAME
Group: FDDI
Type: Indication
Command ID: 0x182

Rx FRAME
Group: FDDI
Type: Indication
Command ID: 0x182

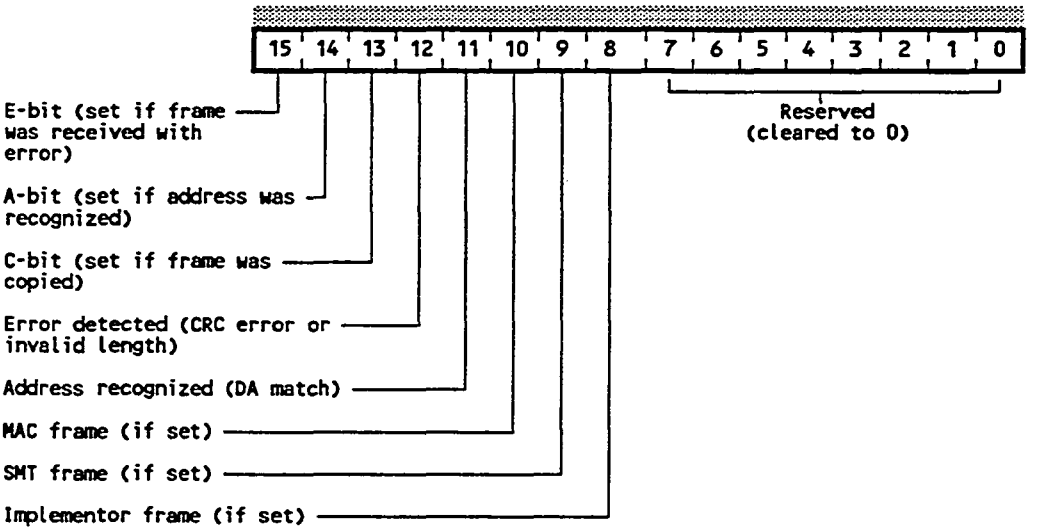


Figure 4–23 . Frame Status Field

• Length in Bytes of Received Frame (2 bytes)

This specifies the total byte length of the received frame. This value includes the 13-byte MAC header and 8-byte LLC header. It does **not** include the 3-byte pad used to longword-align the frame.

• Number of Fragments in Received Frame (2 bytes)

The value is the number of fragments listed in the following list.

• Command Tag for Data Fragment *n* (4 bytes)

This is the command tag of a buffer list element which points to the received data. This tag was supplied by the host to the controller via the Set Up Receive Buffers directive (see p. 4-14).

• Length in Bytes of Data Fragment *n* (4 bytes)

This field contains the length of the data fragment referred to by the corresponding Command Tag parameter.

REPORT STATION ADDRESS

Group: FDDI
Type: Response
Command ID: 0x184

REPORT STATION ADDRESS
Group: FDDI
Type: Response
Command ID: 0x184

The 4211 issues Report Station Address as a result of having received a Request Station Address from the host. This response contains the controller's station address in both Internet and bit transmission format.

The data structure of Report Station Address is shown on p. B-15. Its format is as follows:

Size (in bytes)	DESCRIPTION
8	Command Control Block
4	Command Tag
6	Station Address in Internet Format (Canonical)
6	Station Address in Bit Transmission Format (Non-Canonical)

Figure 4–24 . Report Station Address

FIELDS

The fields in Report Station Address are as follows:

• Command Control Block (8 bytes)

This field contains the Command Control Block, as defined on p. 4-3. The controller sets the command ID field to 0x184 for Report Station Address.

• Command Tag (4 bytes)

This field contains the host-supplied command tag from the Request Station Address command to which the 4211 is responding.

• Station Address, Internet Format (6 bytes)

This field contains the controller's station address, represented using the format specified by Internet Protocol (IP). That is, it is stored byte-wise, with the most significant byte first.

• Station Address, Bit Transmission Format (6 bytes)

This field contains the controller's station address in bit transmission format. That is, it is stored byte-wise, with the least significant byte first.

RING STATUS INDICATION

Group: FDDI
Type: Indication
Command ID: 0x185

RING STATUS INDICATION
Group: FDDI
Type: Indication
Command ID: 0x185

The Ring Status Indication reports a change in the state of the FDDI ring — from "ring down" to "ring up", or vice versa. The 4211 writes this message asynchronously to the default response channel[12]

The 4211 will not issue this indication until the host attempts to connect the station to the ring (see Ring Control directive, p. 4-20). Once this attempt has been made, the board will issue the indication each time it successfully completes a command to connect to (or disconnect from) the ring. It will also issue the indication whenever the ring goes down for any reason.

Since the Ring Status Indication is used to signal a *change* in the ring status, it is not issued if the 4211 fails in an attempt to connect to the ring. Likewise, it is not issued if the host instructs the 4211 to connect or disconnect when the board is already in the specified state. For more information, see "Connecting the Station to the Ring", p. 3-6, and the description of the Ring Control directive on p. 4-20.

The data structure of the Ring Status Indication is shown on p. B-15. Its format is as follows:

Size (in bytes)	DESCRIPTION
8	Command Control Block
4	Ring Status

Figure 4–25 . Ring Status Indication

FIELDS

The fields in the Ring Status Indication are as follows:

• Command Control Block (8 bytes)

This field contains the Command Control Block, as defined on p. 4-3. The controller sets the command ID field to 0x185 for Ring Status Indication.

• Ring Status (4 bytes)

This field contains the status of the ring. Table 4-16 shows the possible status codes.

[12] The default response channel is the initial board-to-host channel created by the host when it boots the RC Interface. The host can detect pending messages by polling the channel and/or enabling interrupts, as discussed at length in Chapter 3 of the *Common Boot and RC Interface User's Guide*.

RING STATUS INDICATION

Group: FDDI
Type: Indication
Command ID: 0x185

RING STATUS INDICATION
Group: FDDI
Type: Indication
Command ID: 0x185

Table 4–16 . Ring Status Codes

STATUS CODE	DESCRIPTION
0x1	Ring up. Station is connected to the network and may transmit and receive frames.
0x2	Ring down. Station is disconnected from ring.

CAM ADDRESS RESPONSE

Group: FDDI
Type: Response
Command ID: 0x187

CAM ADDRESS RESPONSE
Group: FDDI
Type: Response
Command ID: 0x187

The 4211 issues a CAM Address Response as a result of having received either a Set CAM Address or Clear CAM Address request from the host. It reports whether the host was successful in its attempt to modify the CAM.

NOTE: CAM Address Response is not the same as another RC command with a similar name — Report CAM Address. The latter is only issued in response to the Get CAM Address request. For an overview of the CAM-related commands, see p. 3-13.

The data structure of the response is shown on p. B-15. Its format is as follows:

Size (in bytes)	DESCRIPTION
8	Command Control Block
4	Command Tag
4	Status

Figure 4–26 . CAM Address Response

FIELDS

The fields in the CAM Address Response are as follows:

• Command Control Block (8 bytes)

This field contains the Command Control Block, as defined on p. 4-3. The controller sets the command ID field to 0x187 for CAM Address Response.

• Command Tag (4 bytes)

This field contains the host-supplied command tag from the CAM-related command to which the 4211 is responding.

• Status (4 bytes)

This field reports the success or failure of the host's attempt to modify the CAM contents. Possible values are shown in Table 4-17.

CAM ADDRESS RESPONSE

Group: FDDI
Type: Response
Command ID: 0x187

CAM ADDRESS RESPONSE
Group: FDDI
Type: Response
Command ID: 0x187

Table 4–17 . Status Codes in CAM Address Response

CODE	MEANING
0x0	Request successful.
0x1	CAM table full. Request failed.
0x2	CAM address already present when trying to SET. Request failed.
0x3	CAM address not present when trying to CLEAR. Request failed.
0x4	CAM feature not supported on this board version. Request failed.

MIB ATTRIBUTE RESPONSE

Group: FDDI
Type: Response
Command ID: 0x188

MIB ATTRIBUTE RESPONSE
Group: FDDI
Type: Response
Command ID: 0x188

The 4211 issues a MIB Attribute Response in reply to a host-issued Get MIB Attribute request. In order to interpret the data returned by this response, you must be familiar with the TLV parameter structure of MIB attributes. For details, refer to Section 6 ("Services") of the rev. 6.2 SMT specifications.

The data structure of the response is shown on p. B-15. Its format is as follows:

Size (in bytes)	DESCRIPTION
8	Command Control Block
4	ID # of Response Channel
4	Command Tag
4	Attribute Name
4	Data Area Needed for Attribute(s)
4	Attribute Index
N	Attribute Definition (Variable-length)

Figure 4–27 . MIB Attribute Response

FIELDS

The fields in the MIB Attribute Response are as follows:

• Command Control Block (8 bytes)

This field contains the Command Control Block, as defined on p. 4-3. The controller sets the command ID field to 0x188 for MIB Attribute Response.

• ID # of Response Channel (4 bytes)

This field echoes back the "ID # of Response Channel" field from the Set MIB Attribute request to which the 4211 is responding.

• Command Tag (4 bytes)

This field echoes back the host-supplied command tag from the Set MIB Attribute request to which the 4211 is responding.

MIB ATTRIBUTE RESPONSE

Group: FDDI
Type: Response
Command ID: 0x188

MIB ATTRIBUTE RESPONSE
Group: FDDI
Type: Response
Command ID: 0x188

• Attribute Name (4 bytes)

This field contains the Parameter_Type of the attribute or MIB Group supplied by the Get MIB Attribute request.

• Data Area Needed for Attribute(s) (4 bytes)

This field contains a multiple of 0x24 (36 bytes for each attribute being reported). The host does NOT use this value to "unpack" the attribute(s).

• Attribute Index (4 bytes)

This field echoes back the Attribute Index field specified in the request to which the 4211 is responding.

• Attribute Definition (Variable-Length)

This field contains a byte stream with the requested attribute(s). The data is formatted as follows:

1. The data for each attribute is presented *exactly* as defined in the SMT 6.2 specifications, starting with the Parameter_Type and continuing to the end of the attribute definition.

2. If multiple attributes are being returned, the data is packed. No terminator is placed between attributes or at the end of the data.

To extract the data from the Attribute Definition field, use the first eight bytes of the field to identify the Parameter_Type and Parameter_Length of the first attribute. Use the Parameter_Length value to determine where the attribute definition ends. If the controller is only reporting one MIB attribute, this should correspond with end of the MIB Attribute Response, excluding any pad bytes used to longword-align the response.

If board is returning multiple attributes, use each successive Parameter_Length to locate the start of the next attribute definition.

According to 6.2 SMT specifications, attributes are order-independent. Therefore, the host should not assume that the attributes will be presented in the same order that they appear in the SMT specifications.

It is advisable for the host to maintain tables of useful information such as the size of MIB groups and the Parameter_Type and Parameter_Length associated with specific attributes. This information is useful for performing sanity checking.

SMT EVENT INDICATION

Group: FDDI
Type: Indication
Command ID: 0x189

SMT EVENT INDICATION
Group: FDDI
Type: Indication
Command ID: 0x189

If the host has enabled event tracing via the SMT Trace directive (p. 4-35), the 4211 will issue an SMT Event Indication each time it detects an SMT event.

The 4211 writes the SMT Event Indication into the default response channel (i.e. the initial board-to-host channel created at RC boot time).

The data structure of the indication is shown on p. B-15. Its format is as follows:

Size (in bytes)	DESCRIPTION
8	Command Control Block
4	Attribute ID
n	Variable-length SMT Message

Figure 4–28 . SMT Event Indication

FIELDS

The fields in the SMT Event Indication are as follows:

• Command Control Block (8 bytes)

This field contains the Command Control Block, as defined on p. 4-3. The controller sets the command ID field to 0x189 for SMT Event Indication.

• Attribute ID (4 bytes)

This value is the ID of the MIB attribute associated with this event. These IDs are given in the SMT rev. 6.2 specifications. For example, if the controller is reporting an event concerning fddiMACNeighborChange, it writes the value 0x208F to this field.

• Variable-length SMT Message

This field consists of an array of u32's containing the event message. The data in this field matches the SMT 6.2 definition of the MIB attribute. For example, the format of a message concerning the attribute fddiMACNeighborChange is shown on page 92 of the 6.2 SMT document.

REPORT HARDWARE PARAMETERS

Group: FDDI
Type: Response
Command ID: 0x18A

REPORT HARDWARE PARAMETERS
Group: FDDI
Type: Response
Command ID: 0x18A

Report Hardware Parameters is issued in response to a Get Hardware Parameters request (p. 4-41). It reports the current mode of operation of the hardware registers specified in the request.

The format of response is as follows:

SIZE (in bytes)	DESCRIPTION
8	Command Control Block
4	Command Tag
4	Parameter Count
2	Parameter Type
2	Parameter Data
2	Parameter Type
2	Parameter Data
⋮	⋮
2	Parameter Type
2	Parameter Data

Figure 4–29 . Report Hardware Parameters Response

FIELDS

The fields in the Report Hardware Parameters response are as follows:

• Command Control Block (8 bytes)

This field contains the Command Control Block, as defined on p. 4-3. The host sets the command ID to 0x18A for Report Hardware Parameters.

• Command Tag (4 bytes)

This field contains the host-supplied command tag from the command to which the 4211 is responding.

• Parameter Count (4 bytes)

This field contains the number of parameter reports supplied in the remainder of this command. For each parameter being reported, there is one pair of Parameter Type and Parameter Data fields. Example: If this field contains 0x5, there are five pairs of Parameter Type/Parameter Data fields in the remainder of the response.

REPORT HARDWARE PARAMETERS

Group: FDDI
Type: Response
Command ID: 0x18A

REPORT HARDWARE PARAMETERS
Group: FDDI
Type: Response
Command ID: 0x18A

• Parameter Type *n* (2 bytes)

The code in this field identifies the type of parameter being reported. Table 4-18 lists the codes for the available registers and functions.

Table 4-18 . Parameter Types for Report Hardware Parameters

CODE	PARAMETER TYPE	VALID PARAMETER DATA
0x0001	CAM Command Register	See Diagram 4-4, p. 4-60
0x0002	Front End Command Register	See Diagram 4-5, p. 4-61
0x0003	(reserved)	------
0x0004	Set Formac Flush Mode	See Table 4-19, p. 4-62
0x0005	Disable/Enable FCS Generation	See Table 4-20, p. 4-62
0x0006	Optical Bypass Relay Control	See Table 4-21, p. 4-63

• Parameter Data *n* (2 bytes)

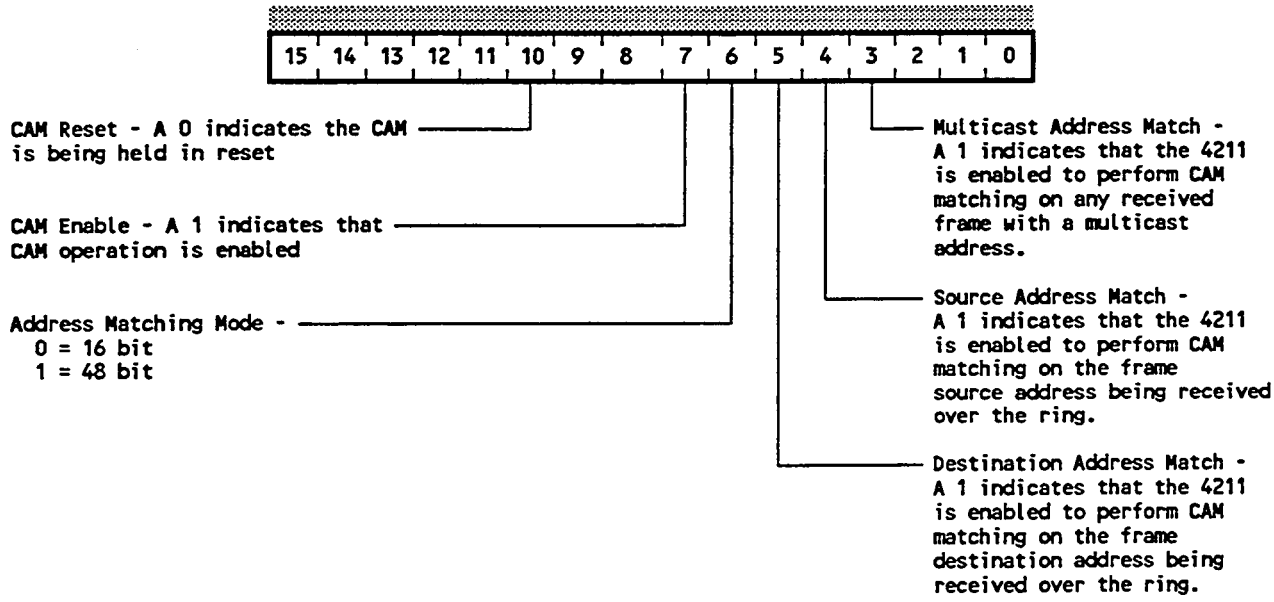
This field contains either: 1) a mapped field of control bits written to a specific register, or 2) a code indicating the mode of a specific function. The data written to this field depends on the Parameter Type field with which it is paired (i.e. the 2-byte Parameter Type field immediately preceding it). The following five diagrams and tables describe the data you may expect to see for each of the five parameter types listed in Table 4-18.

REPORT HARDWARE PARAMETERS

Group: FDDI
Type: Response
Command ID: 0x18A

REPORT HARDWARE PARAMETERS

Group: FDDI
Type: Response
Command ID: 0x18A



NOTE: Bits which are not referenced above are reserved. Their values are always 0x0.

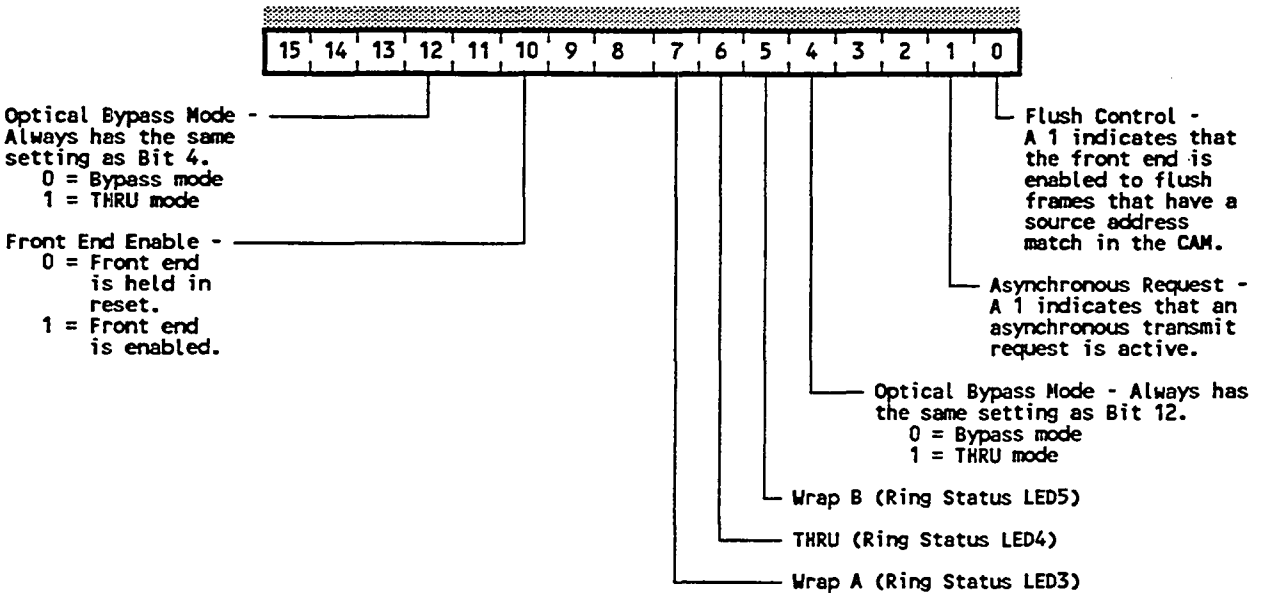
Diagram 4-4 . Returned Data in CAM Command Register

REPORT HARDWARE PARAMETERS

Group: FDDI
Type: Response
Command ID: 0x18A

REPORT HARDWARE PARAMETERS

Group: FDDI
Type: Response
Command ID: 0x18A



NOTE: Bits which are not referenced above are reserved. Their values are always 0x0.

Diagram 4–5 . Returned Data in Front End Command Register

Table 4–19 . Returned FORMAC Frame Flush Modes

CODE	FORMAC FRAME FLUSH MODE ENABLED
0x0	Flush when no destination match
0x1	No flush on destination or source address matches
0x2	No flush (promiscuous mode)

REPORT HARDWARE PARAMETERS

Group: FDDI
Type: Response
Command ID: 0x18A

REPORT HARDWARE PARAMETERS
Group: FDDI
Type: Response
Command ID: 0x18A

Table 4–20 . Returned FCS Generation Modes

CODE	MODE
0x0	FCS is enabled. Frame Check Sequence is sent on transmit frames.
0x1	FCS is disabled.

Table 4–21 . Returned Optical Bypass Control Modes

CODE	MODE
0x0	Optical Bypass Relay not present
0x1	Optical Bypass Relay present and bypass mode enabled
0x2	Optical Bypass Relay present and THRU mode enabled

REPORT CAM ADDRESS

Group: FDDI
Type: Response
Command ID: 0x18B

REPORT CAM ADDRESS

Group: FDDI
Type: Response
Command ID: 0x18B

The 4211 issues Report CAM Address in response to a host-issued Get CAM Address request (p. 4-43). If the CAM is not present, the return status indicates "no CAM present" and a zero address field. If the CAM is present, the 4211 returns a "success" and the 48-bit MAC address stored at the requested location.

NOTE: Report CAM Address is not the same as another RC command with a similar name — CAM Address Response. The latter is only issued in response to the Set CAM Address and Clear CAM Address requests. For an overview of the CAM-related commands, see p. 3-13.

The format of the response is as follows:

Size (in bytes)	DESCRIPTION
8	Command Control Block
4	Command Tag
2	Status
6	MAC Address

Figure 4–30 . Report CAM Address

FIELDS

The fields in Report CAM Address are as follows:

• Command Control Block (8 bytes)

This field contains the Command Control Block, as defined on p. 4-3. The controller sets the command ID field to 0x18B for Report CAM Address.

• Command Tag (4 bytes)

This field contains the host-supplied command tag from the CAM-related command to which the 4211 is responding.

• Status (2 bytes)

This field reports the success or failure of the host's attempt to read the CAM entry. Possible values are shown in Table 4-22.

REPORT CAM ADDRESS

Group: FDDI
Type: Response
Command ID: 0x18B

REPORT CAM ADDRESS

Group: FDDI
Type: Response
Command ID: 0x18B

Table 4–22 . Status Codes in Report CAM Address

CODE	MEANING
0x0	CAM is present. Query successful.
0x4	CAM is not present. Query failed.

- **MAC Address (6 bytes)**

  If the Status field indicates that the CAM is present, this field contains the 48-bit MAC address stored at the requested CAM location. If no CAM is present, this field is cleared to 0.

CHAPTER 5
SPECIFICATIONS

ARCHITECTURE

Processor	AMD Am29000 RISC
Network Interface	AMD SUPERNET™ chipset
Data Communications Buffer	1 Mbyte

VMEbus SPECIFICATIONS

- VMEbus data transfers in excess of 30 Mbytes/sec.
- Supports DMA word aligned longword and word transfers. No unaligned byte transfer capability.
- **Options (VMEbus Master):**

Addressing	16-, 24-, and 32-bit addressing 16-, and 32-bit data transfers
Address modifiers	all options
Requester options	any one of R(0), R(1), R(2), R(3) (jumper-selectable)
RWD (Release When Done)	

- **Options (VMEbus Slave):**

Addressing	A16:D16
Address Modifiers	0x29, 0x2D
Interrupter options	any one or IR(x-y) where 1 <= x <= 7 and 1 <= y <= 7 D08 any vector

Dual priority interrupt controllers (Am9519A) provide a unique interrupt vector for each individual interrupt source.

FDDI SPECIFICATIONS

Configurations	Single PHY single MAC (SASSM) Dual PHY single MAC (DASSM) Hardware support for dual PHY, dual MAC (DASDM) using two 4211 controllers
SMT	FDDI Station Management standards, Rev. 6.2
Optical Bypass	I/O port with up to 300 mA current sink for control of external optical bypass switch
Connectors	ST™-compatible optical connectors 6-pin mini-DIN optical bypass control

TIMER DEFAULTS

The default value of TVX is 2.5 milliseconds.
The default for T_req is 165 milliseconds.
The value of T_Min is 4 msec., and T_Max is 165 msec.

POWER REQUIREMENTS

• V/FDDI 4211	5.8 amps typical at +5 V_{DC} / 25 degrees C

NOTE: In hardware variation levels H04211-008,REVxxx and earlier, the power requirements are:
7.8 amps typical @ +5V_{DC} / 25 degrees C
1 amp typical @ +12V_{DC} / 25 degrees C

MECHANICAL (nominal)

The 4211 occupies one 6U single-height VMEbus slot.

Width:	160 mm
Length:	233 mm
Thickness:	20 mm
Weight:	566 grams

OPERATING ENVIRONMENT

Temperature	0 to 55 degrees C
Relative Humidity	5% to 95% noncondensing
Altitude	-1000 to 15,000 feet
Air Flow	250 LFM minimum

STORAGE ENVIRONMENT

Storage Temperature	0 to 125 degrees C
Storage Humidity	5% to 95% noncondensing
Storage Altitude	-1000 to 50,000 feet

LEDs

The 4211 has nine LEDs — labelled LED1 through LED9 in the board drawing on p. 2-2.

LED1 and LED2 are mutually exclusive (i.e. when one turns on, the other turns off). When the board is powered-up or reset, the LEDs will toggle on and off for about 10 seconds while the 4211 performs its power-up diagnostics. Once the 4211's firmware has booted up and the Common Boot interface is active, the red LED (LED1) turns off the green LED (LED2) turns on. Then, once the host boots the RC Interface, the LEDs return to toggling on/off again.

LED3, LED4, and LED5 indicate the state of the ring. LED4 signals THRU, LED3 signals the status of the primary ring, and LED5 signals the status of the secondary ring. If the THRU bit is set (i.e. LED4 is lit), one of the other bits indicates the THRU condition. If the THRU bit is not set, the other bits indicate WRAP condition. All bits off indicates a "ring down" condition. Table 5-1 summarizes all possible ring conditions which can be signalled by these LEDs. These ring states are defined in section 9.7.6.2 of the SMT rev. 6.2 specifications.

Table 5–1 . Ring State LEDs (LED3 - LED5)

RING STATE	PRI LED3	THRU LED4	SEC LED5
Isolated	OFF	OFF	OFF
Wrap_A	ON	OFF	OFF
Wrap_B	OFF	OFF	ON
Wrap_AB	ON	OFF	ON
Thru_A	ON	ON	OFF
Thru_B	OFF	ON	ON
Thru_AB	ON	ON	ON

LED6, LED7, LED8, and LED9 indicate the status of the 4211's four DMA channels. LED6 and LED7 signal transmit activity when the ring is up. LED8 and LED9 signal when the channel is programmed and the board is ready to receive a new frame. These LEDs are summarized in Table 5-2.

Table 5–2 . DMA Status LEDs (LED6 - LED9)

Channel	LED	Function
Tx (Synchronous)	LED6	ON = transmitting; OFF = not active
Tx (Asynchronous)	LED7	ON = transmitting; OFF = not active
Rx Channel A	LED8	ON = ready to receive; OFF = not ready
Rx Channel B	LED9	ON = ready to receive; OFF = not ready

NOTE: LEDs 3 through 9 are only supported on controllers with hardware revision level HO4211-xxx,REVD and later. LED1 and LED2 are supported on all versions of the controller.

DEFAULT REPORT/COMMAND PARAMETERS

This section states the default values of parameters which can be adjusted by certain members of the generic RC command set. The generic RC commands referred to below are documented in the *Common Boot and RC Interface User's Guide.*

Default Size of DMA Transfers

The 4211 supports 16- and 32-bit DMA transfers and has an 8-bit transfer counter. The default transfer count is 256 transfers per burst, which is the maximum supported by the counter. Using this default setting, the DMA burst sizes are as follows:

Burst size (16-bit transfers) = 512 bytes per burst

Burst size (32-bit transfers) = 1024 bytes per burst

The above burst sizes can be adjusted using the Set DMA Burst directive.

Default Bus Timeout

The 4211's default bus timeout is infinite. To set an explicit timeout value, issue the Set Bus Timeout directive.

Error Messages

The generic RC command set includes two indications which can be used by the controller to pass error messages to the host. These indications are issued asynchronously by the 4211 to the default response channel (i.e. the initial board-to-host channel created at RC boot time). The two indications are:

- **Error Message (Command ID 0x82)**

  No 4211-specific errors are currently defined for the Error Message indication. For information on the generic error messages, refer to the command description in the *Common Boot & RC Interface User's Guide.*

- **Report Printf Call (Command ID 0x81)**

  Report Printf Call is used to report Request Denied Frames (RDFs) or Parameter Management Frames (PMF) to the host. This occurs when the 4211 detects an error while attempting to execute a Set MIB Attribute directive. The "Variable-Length ASCII String" field in the Report Printf Call indication contains a code which identifies the RDF or PMF. Table 5-3 lists the error codes which may appear in this field.

  The Set MIB Attribute directive, which is described on p. 4-26, is issued by the host to change the value of an attributes in the 4211's Management Information Base.

Table 5–3 . 4211-Specific Messages Returned via Report Printf Call

Reason Code	Description
0x01	Frame Class not supported (RDF)
0x02	Frame Version not supported (RDF)
0x04	BadSetCount (PMF) - Indicates that the value of SetCount used did not match the Responder's current value.
0x05	ReadOnly (PMF) - Attempted to change a read-only parameter.
0x06	NoParam (PMF) - The requested parameter is not supported by this station.
0x07	NoMore (PMF) - Returned in response to an Add or Remove when there is no more room for Adds, or no more values to Remove.
0x08	Out_of_range - Value given in Change or Add or Remove is out of bounds.
0x09	NotAuthorized (PMF) - Authorization field does not match or station does not otherwise permit the requested operation.
0x0A	Length Error (RDF, PMF) - Occurs because one of the parameters in the message body had an incorrect length for the parameter type or because the reported total InfoField_length did not correlate with the total length of info field contents received.
0x0B	FrameTooLong (PMF) - Occurs when a Response would be greater than the maximum allowed frame size.
0x0C	IllegalParameter (PMF) - Occurs when an unrecognized parameter is present. (One not in the SMT specification)

For more information, refer to the command description of Report Printf Call in the *Common Boot & RC Interface User's Guide*.

CONNECTOR PINOUTS

J1 Connector (Daughter Card I/O)

The following table shows the pinout of the J1 connector on the 4211. A brief description of the signals appears after the table.

Table 5–4 . J1 - Daughter Card I/O

Signal	Pin	I/O	Signal	Pin	I/O
PA0	J1-1	I	PA1	J1-2	I
PA2	J1-3	I	PA3	J1-4	I
PA4	J1-5	I	PA5	J1-6	I
PA6	J1-7	I	PA7	J1-8	I
PA8	J1-9	I	GND	J1-10	GROUND
PA9	J1-11	I	PA10	J1-12	I
PA11	J1-13	I	PA12	J1-14	I
PA13	J1-15	I	PA14	J1-16	I
PA15	J1-17	I	PA16	J1-18	I
PA17	J1-19	I	PA18	J1-20	I
PA19	J1-21	I	PA20	J1-22	I
PA21	J1-23	I	PA22	J1-24	I
PA23	J1-25	I	PD0	J1-26	I/O
PD1	J1-27	I/O	PD2	J1-28	I/O
PD3	J1-29	I/O	VCC	J1-30	+5V
PD4	J1-31	I/O	PD5	J1-32	I/O
PD6	J1-33	I/O	PD7	J1-34	I/O
PD8	J1-35	I/O	PD9	J1-36	I/O
PD10	J1-37	I/O	PD11	J1-38	I/O
PD12	J1-39	I/O	PD13	J1-40	I/O
PD14	J1-41	I/O	PD15	J1-42	I/O
PD16	J1-43	I/O	PD17	J1-44	I/O
PD18	J1-45	I/O	PD19	J1-46	I/O
PD20	J1-47	I/O	PD21	J1-48	I/O
PD22	J1-49	I/O	GND	J1-50	GROUND
PD23	J1-51	I/O	PD24	J1-52	I/O
PD25	J1-53	I/O	PD26	J1-54	I/O
PD27	J1-55	I/O	PD28	J1-56	I/O
PD29	J1-57	I/O	PD30	J1-58	I/O
PD31	J1-59	I/O	BOK	J1-60	O

Signal(s)	Pin(s)	Description
PA0 - PA8	J1-1 - J1-9	29K address bus lines 0 - 8
GND	J1-10	Ground
PA9 - PA23	J1-11 - J1-25	29K address bus lines 9 - 23
PD0 - PD3	J1-26 - J1-29	29K data bus lines 0 - 3
VCC	J1-30	Power (+5V)
PD4 - PD22	J1-31 - J1-49	29K data bus lines 4 - 22
GND	J1-50	Ground
PD23 - PD31	J1-51 - J1-59)	29K data bus lines 23 - 31
BOK	J1-60	Board OK; indicates whether or not the 4211 firmware has booted successfully. This signal controls both the red and green LED.

J2 Connector (Daughter Card I/O)

The following table shows the pinout of the J2 connector on the 4211. A brief description of the signals appears after the table.

Table 5–5 . J2 - Daughter Card I/O

Signal	Pin	I/O	Signal	Pin	I/O
2X_SYSCLK	J2-1	I	SYSCLK	J2-2	I
DMACLK	J2-3	I	1_4_SYSCLK	J2-4	I
~WR	J2-5	I	~BINV	J2-6	I
~EBREQ	J2-7	I	~BUFEN	J2-8	I
~BUFREQ	J2-9	I	GND	J2-10	GROUND
~DMACS	J2-11	I	~REFREQ	J2-12	I
~CLKSTP	J2-13	O	~BUFRDY	J2-14	O
~REFACK	J2-15	O	~AIORDY	J2-16	O
~SYNCRST	J2-17	O	~VMEINT	J2-18	O
OPT0	J2-19	I	OPT1	J2-20	I
~ACFAIL	J2-21	O	~FOR_CNT_INT	J2-22	O
~RX_INT	J2-23	O	~TXS_INT	J2-24	O
~TXA_INT	J2-25	O	~DMAINT	J2-26	O
~MBINT	J2-27	O	BUSERR	J2-28	O
(not used)	J2-29		VCC	J2-30	+5V
~STRT23	J2-31	I	~STRT01	J2-32	I
~BUSRST	J2-33	I	~SVINT	J2-34	I
~REQBUS	J2-35	I	~CLRMBINT	J2-36	I
~LDVME	J2-37	I	~LDAM	J2-38	I
~TX_STAT	J2-39	I	~CAM	J2-40	I
~PHY2	J2-41	I	~CNT1_REG	J2-42	I
~9513	J2-43	I	~FE_CMD_STAT	J2-44	I
~ENDEC	J2-45	I	~FORMAC	J2-46	I
~LDVADDR	J2-47	I	SYSFAIL	J2-48	I
DS0EN	J2-49	I	GND	J2-50	GROUND
DS1EN	J2-51	I	IPL2	J2-52	I
IPL1	J2-53	I	IPL0	J2-54	I
~SET	J2-55	I	~DISBUMP	J2-56	I
~SEQ	J2-57	I	ENBURST	J2-58	I
BUSWR	J2-59	I	32BIT	J2-60	I

Signal(s)	Pin(s)	Description
2X_SYSCLK	J2-1	Twice system (29K) clock
SYSCLK	J2-2	System (29K) clock
DMACLK	J2-3	DMA controller clock (same frequency as SYSCLK)
1_4_SYSCLK	J2-4	One-fourth system (29K) clock *frequency
~WR	J2-5	29K data write line
~BINV	J2-6	29K bus invalid
~EBREQ	J2-7	Early 29K buffer RAM request
~BUFEN	J2-8	Data buffer enable for buffer RAM reads
~BUFREQ	J2-9	29K buffer RAM request
GND	J2-10	Ground
~DMACS	J2-11	DMA controller register select strobe
~REFREQ	J2-12	Buffer RAM refresh cycle request
~CLKSTP	J2-13	DMA controller clock freeze signal
~BUFRDY	J2-14	29K buffer RAM access ready line
~REFACK	J2-15	Buffer RAM refresh cycle acknowledge strobe

Signal(s)	Pin(s)	Description
~AIORDY	J2-16	29K asynchronous I/O access ready line
~SYNCRST	J2-17	Synchronized slave or VME reset
~VMEINT	J2-18	VME interrupt
OPT0 - OPT1	J2-19 - J2-20	29K transfer size (byte, word, longword)
~ACFAIL	J2-21	VME ACFAIL interrupt
~FOR_CMT_INT	J2-22	Front end FORMAC/CMT interrupt
~RX_INT	J2-23	Front end receive interrupt
~TXS_INT	J2-24	Front end synchronous transmit interrupt
~TXA_INT	J2-25	Front end asynchronous transmit interrupt
~DMAINT	J2-26	VME master DMA termination interrupt
~MBINT	J2-27	VME slave mail box interrupt
BUSERR	J2-28	Input VME master bus error status
VCC	J2-30	Power (+5V)
~STRT23	J2-31	VME master strobe for starting at Data Word 23
~STRT01	J2-32	VME master strobe for starting at Data Word 01
~BUSRST	J2-33	Reset strobe for Buspacket hardware
~SVINT	J2-34	Set VME interrupt strobe
~REQBUS	J2-35	VME master bus request strobe
~CLRMBINT	J2-36	Clear VME slave mail box interrupt strobe
~LDVME	J2-37	VME master load transfer/burst count strobe
~LDAM	J2-38	VME master load address modifier strobe
~TX_STAT	J2-39	Front end transmit completion status
~CAM	J2-40	Content addressable memory R/W strobe
~PHY2	J2-41	Secondary front end encoder/decoder R/W strobe
~CMT1_REG	J2-42	Front end strobe for static signal latch / input port
~9513	J2-43	Front end SMT counter strobe
~FE_CMD_STAT	J2-44	Front end status register strobe
~ENDEC	J2-45	Primary front end encoder/decoder R/W strobe
~FORMAC	J2-46	Front end media access controller R/W strobe
~LDVADDR	J2-47	VME master load starting bus address strobe
SYSFAIL	J2-48	Static VME system fail enable
DS0EN	J2-49	VME master Data Strobe 0 enable
GND	J2-50	Ground
DS1EN	J2-51	VME master Data Strobe 1 enable
IPL2	J2-52	Most significant bit of VME interrupt level (7-1)

Signal(s)	Pin(s)	Description
IPL1	J2-53	Second most significant bit of VME interrupt level
IPL0	J2-54	Least significant bit of VME interrupt level
~SET	J2-55	Static VME master force VA1 to a "1"
~DISBUMP	J2-56	Static VME master freeze address enable
~SEQ	J2-57	Static VME master block mode enable
ENBURST	J2-58	Static VME master bus burst enable
BUSWR	J2-59	Static VME master bus write enable
32BIT	J2-60	Static VME master longword enable

J3 Connector (Configuration Multiplexer Input)

The following table shows the pinout of the J3 connector on the 4211. A brief description of the signals appears after the table.

Table 5–6 . J3 - Configuration Multiplexer Input

Signal	Pin	I/O	Signal	Pin	I/O
MI9	J3-1	I	MI8	J3-2	I
MI7	J3-3	I	MI6	J3-4	I
MI5	J3-5	I	MI4	J3-6	I
MI3	J3-7	I	MI2	J3-8	I
MI1	J3-9	I	MI0	J3-10	I
GND	J3-11	GROUND	LOCAL_CLK_PLUS	J3-12	I/O
LOCAL_CLK_MINUS	J3-13	I/O	~MAC2_PRES	J3-14	I

Signal(s)	Pin(s)	Description
MI9	J3-1	Multiplexer input line 9
MI8	J3-2	" " " 8
MI7	J3-3	" " " 7
MI6	J3-4	" " " 6
MI5	J3-5	" " " 5
MI4	J3-6	" " " 4
MI3	J3-7	" " " 3
MI2	J3-8	" " " 2
MI1	J3-9	" " " 1
MI0	J3-10	" " " 0
GND	J3-11	Ground
LOCAL_CLK_PLUS	J3-12	Local clock
LOCAL_CLK_MINUS	J3-13	Local clock
GND	J3-14	Ground

J4 Connector (Configuration Multiplexer Output)

The following table shows the pinout of the J4 connector on the 4211. A brief description of the signals appears after the table.

Table 5–7 . J4 - Configuration Multiplexer Output

Signal	Pin	I/O	Signal	Pin	I/O
MO9	J4-1	O	MO8	J4-2	O
MO7	J4-3	O	MO6	J4-4	O
MO5	J4-5	O	MO4	J4-6	O
MO3	J4-7	O	MO2	J4-8	O
MO1	J4-9	O	MO0	J4-10	O
GND	J4-11	GROUND	LOCAL_CLK_PLUS	J4-12	I/O
LOCAL_CLK_MINUS	J4-13	I/O	~MAC2_PRES	J4-14	O

Signal(s)	Pin(s)	Description
MO9	J4-1	Multiplexer output line 9
MO8	J4-2	" " " 8
MO7	J4-3	" " " 7
MO6	J4-4	" " " 6
MO5	J4-5	" " " 5
MO4	J4-6	" " " 4
MO3	J4-7	" " " 3
MO2	J4-8	" " " 2
MO1	J4-9	" " " 1
MO0	J4-10	" " " 0
GND	J4-11	Ground
LOCAL_CLK_PLUS	J4-12	Local clock
LOCAL_CLK_MINUS	J4-13	Local clock
GND	J4-14	Ground

J5 Connector (Companion Board I/O)

The following table shows the pinout of the J5 connector on the 4211. A brief description of the signals appears after the table.

Table 5–8 . J5 - Companion Board I/O

Signal	Pin	I/O	Signal	Pin	I/O
ENDEC1_S0	J5-1	O	ENDEC1_S1	J5-2	O
ENDEC1_S2	J5-3	O	ENDEC2_S0	J5-4	O
ENDEC2_S1	J5-5	O	ENDEC2_S2	J5-6	O
~PHY2	J5-7	O	~9513	J5-8	O
LQME1	J5-9	O	GND	J5-10	GROUND
LQME2	J5-11	O	1_4_SYSCLK	J5-12	O
~FSVLD	J5-13	O	~XSAMAT	J5-14	I
SDRCVD	J5-15	O	~XDAMAT	J5-16	I
TOKCNT	J5-17	O	LNGADR	J5-18	O
~BWR	J5-19	O	VCC	J5-20	+5V
~AIORDY	J5-21	I	~FOR_CMT_INT	J5-22	I
~CMT1_INTR	J5-23	I	~CMT2_INTR	J5-24	I
~9513_1INTR	J5-25	I	~9513_2INTR	J5-26	I
~CMT1_REG	J5-27	O	BCLK3	J5-28	O
ERRINC	J5-29	O	GND	J5-30	GROUND
~CAM	J5-31	O	~SYNCRST	J5-32	O
~MISFRM	J5-33	O		J5-34	

Signal(s)	Pin(s)	Description
ENDEC1_S0	J5-1	Status line 0 on the ENDEC of the primary PHY
ENDEC1_S1	J5-2	Status line 1 on the ENDEC of the primary PHY
ENDEC1_S2	J5-3	Status line 2 on the ENDEC of the primary PHY
ENDEC2_S0	J5-4	Status line 0 on the ENDEC of the secondary PHY
ENDEC2_S1	J5-5	Status line 1 on the ENDEC of the secondary PHY
ENDEC2_S2	J5-6	Status line 2 on the ENDEC of the secondary PHY
~PHY2	J5-7	Multiple chip select for second PHY (selects secondary ENDEC, 9513, CMT register)
~9513	J5-8	Chip select for primary 9513 counter
LQME1	J5-9	Link quality monitor enable for primary ENDEC
GND	J5-10	Ground
LQME2	J5-11	Link quality monitor enable for secondary ENDEC
1_4_SYSCLK	J5-12	One-fourth system (29K) clock frequency
~FSVLD	J5-13	Frame status valid; counts # of frames stripped from ring
~XSAMAT	J5-14	External source address; generated by the CAM when it detects a match when source frame stripping
SDRCVD	J5-15	Start delimiter receive signal; signals when 4211 is starting to receive a frame
~XDAMAT	J5-16	External destination address; generated by the CAM when it detects a match when destination address matching
TOKCNT	J5-17	Token count; number of tokens on ring
LNGADR	J5-18	Long address detection for incoming frames; used in conjunction with SDRCVD (J5-15)
~BWR	J5-19	Buffered write signal; data write line
VCC	J5-20	Power (+5V)
~AIORDY	J5-21	Asynchronous I/O ready line

Signal(s)	Pin(s)	Description
~FOR_CMT_INT	J5-22	Front end FORMAC/CMT interrupt
~CMT1_INTR	J5-23	Interrupt generated by primary CMT; wire OR'ed with ~CMT2_INTR
~CMT2_INTR	J5-24	Interrupt generated by secondary CMT; wire OR'ed with ~CMT1_INTR
~9513_1INTR	J5-25	Interrupt generated by primary 9513 counter; wire OR'ed with ~9513_2INTR
~9513_2INTR	J5-26	Interrupt generated by secondary 9513 counter; wire OR'ed with ~9513_1INTR
~CMT1_REG	J5-27	Chip select for primary CMT register
BCLK3	J5-28	Byte Clock 3 (12.5 MHz)
ERRINC	J5-29	Error increment; a MAC statistic from the FORMAC
GND	J5-30	Ground
~CAM	J5-31	Content-addressable memory chip select
~SYNCRST	J5-32	Synchronous reset signal; generated whenever 29K CPU is reset
~MISFRM	J5-33	Misframe; 4211 was unable to copy frame into memory

J6 Connector (Companion Board I/O))

The following table shows the pinout of the J6 connector on the 4211. A brief description of the signals appears after the table.

Table 5–9 . J6 - Companion Board I/O

Signal	Pin	I/O	Signal	Pin	I/O
BPD0	J6-1	I/O	BPD1	J6-2	I/O
BPD2	J6-3	I/O	BPD3	J6-4	I/O
BPD4	J6-5	I/O	BPD5	J6-6	I/O
BPD6	J6-7	I/O	BPD7	J6-8	I/O
BPD8	J6-9	I/O	GND	J6-10	GROUND
BPD9	J6-11	I/O	BPD10	J6-12	I/O
BPD11	J6-13	I/O	BPD12	J6-14	I/O
BPD13	J6-15	I/O	BPD14	J6-16	I/O
BPD15	J6-17	I/O	BPA2	J6-18	O
BPA3	J6-19	O	VCC	J6-20	+5V
BPA4	J6-21	O	BPA5	J6-22	O
YR0	J6-23	O	YR1	J6-24	O
YR2	J6-25	O	YR3	J6-26	O
YR4	J6-27	O	YR5	J6-28	O
YR6	J6-29	O	GND	J6-30	GROUND
YR7	J6-31	O	FRINC	J6-32	O
LSTINC	J6-33	O		J6-34	

Signal(s)	Pin(s)	Description
BPD0 - BPD8	J6-1 - J6-9	Buffered processor data bus lines 0 - 8
GND	J6-10	Ground
BPD9 - BPD15	J6-11 - J6-17	Buffered processor data bus lines 9 - 15
BPA2 - BPA3	J6-18 - J6-19	Buffered processor address bus lines 2 - 3
VCC	J6-20	Power (+5V)
BPA4 - BPA5	J6-21 - J6-22	Buffered processor address bus lines 4 - 5
YR0 - YR6	J6-23 - J6-29	FORMAC receive bus lines 0 - 6
GND	J6-30	Ground
YR7	J6-31	FORMAC receive bus line 7
FRINC	J6-32	Frame increment; FORMAC signal used to generate MAC statistics
LSTINC	J6-33	Lost frame increment; FORMAC signal used to generate MAC statistics

J7 Connector (Optical Bypass Control)

The following table shows the pinout of the J7 connector on the 4211. A brief description of the signals appears after the table.

Table 5–10 . J7 - Optical Bypass Control

Signal	Pin	I/O	Signal	Pin	I/O
FUSED_VCC	J7-1	+5VF	FUSED_VCC	J7-2	+5VF
~P_OBC_RET	J7-3	O	~S_OBC_RET	J7-4	O
~OBC_PRES	J7-5	I	GND	J7-6	GROUND
CGND	J7-99	(CASE GROUND)			

Signal(s)	Pin(s)	Description
FUSED_VCC	J7-1 and J7-2	Power (+5V) for the 2 relays in the optical bypass switch
~P_OBC_RET	J7-3	Primary optical bypass return line
~S_OBC_RET	J7-4	Secondary optical bypass return line
~OBC_PRES	J7-5	Optical bypass control present; used by software detect when there is an optical bypass switch connected to the 4211
GND	J7-6	Ground
CGND	J7-99	Chassis ground

P1 and P2 Connectors (VMEbus)

All versions of the 4211 have the same VME P1 connector configuration. See Table 5-11 for a list of the P1 signals.

Some versions of the 4211 utilize only row B on the P2 connector. (See Table 5-12.) Others use all three rows on the P2 connector. (See Table 5-13.) The easiest way to determine which board version you have is to look at the P2 connector on your board. If it has one row of pins (32 pins), it uses P2 row B only. If it has three rows of pins (96 pins), it uses P2 rows A and C as well.

Table 5–11 . P1 Connector - VMEbus

Pin	Row A Signal Mnemonic	Row B Signal Mnemonic	Row C Signal Mnemonic
1	D00	BBSY*	D08
2	D01	BCLR*	D09
3	D02	ACFAIL*	D10
4	D03	BGOIN*	D11
5	D04	BGOOUT*	D12
6	D05	BG1IN*	D13
7	D06	BG1OUT*	D14
8	D07	BG2IN*	D15
9	GND	BG2OUT*	GND
10	SYSCLK	BG3IN*	SYSFAIL*
11	GND	BG3OUT*	BERR*
12	DS1*	BRO*	SYSRESET*
13	DS0*	BR1*	LWORD*
14	WRITE*	BR2*	AM5
15	GND	BR3*	A23
16	DTACK*	AMO	A22
17	GND	AM1	A21
18	AS*	AM2	A20
19	GND	AM3	A19
20	IACK*	GND	A18
21	IACKIN*	SERCLK	A17
22	IACKOUT*	SERDAT*	A16
23	AM4	GND	A15
24	A07	IRQ7*	A14
25	A06	IRQ6*	A13
26	A05	IRQ5*	A12
27	A04	IRQ4*	A11
28	A03	IRQ3*	A10
29	A02	IRQ2*	A09
30	A01	IRQ1*	A08
31	-12V	+5VSTDBY	+12V
32	+5V	+5V	+5V

For a description of the signals on the P1 connector, refer to the VMEbus specifications.

Table 5–12 . P2 Connector - VMEbus (P2 Row B Only)

Pin	Row A Signal Mnemonic	Row B Signal Mnemonic	Row C Signal Mnemonic
1		+5V	
2		GND	
3		RESERVED	
4		A24	
5		A25	
6		A26	
7		A27	
8		A28	
9		A29	
10		A30	
11		A31	
12		GND	
13		+5V	
14		D16	
15		D17	
16		D18	
17		D19	
18		D20	
19		D21	
20		D22	
21		D23	
22		GND	
23		D24	
24		D25	
25		D26	
26		D27	
27		D28	
28		D29	
29		D30	
30		D31	
31		GND	
32		+5V	

NOTE: If a signal is not referenced, it is not used.

Table 5–13 . P2 Connector - VMEbus (P2 Rows A, B, and C)

Pin	Row A Signal Mnemonic	Row B Signal Mnemonic	Row C Signal Mnemonic
1	BCLK4	+5V	XMT0
2	~ENDEC	GND	XMT1
3	~ENDEC2	RESERVED	XMT2
4	~BWR	A24	XMT3
5	~AIORDY	A25	XMT4
6	~SYNCRST	A26	XMT5
7	~FOR_CMT_INT	A27	XMT6
8	~PHYT_INT	A28	XMT7
9	BPA2	A29	XMT8
10	BPA3	A30	XMT9
11	BPA4	A31	N/C
12	BPA5	GND	PRI_RCV0
13	OBC1	+5V	PRI~RCV1
14	OBC2	D16	PRI~RCV2
15	DATTACH	D17	PRI~RCV3
16	BPD8	D18	PRI~RCV4
17	BPD9	D19	PRI~RCV5
18	BPD10	D20	PRI~RCV6
19	BPD11	D21	PRI~RCV7
20	BPD12	D22	PRI~RCV8
21	BPD13	D23	PRI~RCV9
22	BPD14	GND	N/C~
23	BPD15	D24	SEC_RCV0
24	~PHY2_INT	D25	SEC~RCV1
25	ENDEC1~S0	D26	SEC~RCV2
26	ENDEC1~S1	D27	SEC~RCV3
27	ENDEC1~S2	D28	SEC~RCV4
28	ENDEC2~S0	D29	SEC~RCV5
29	ENDEC2~S1	D30	SEC~RCV6
30	ENDEC2~S2	D31	SEC~RCV7
31	~OBC_PRES	GND	SEC~RCV8
32	12VN~	+5V	SEC_RCV9

The following is a brief description of the user-defined P2 row A and C signals. For a description of the VME-standard signals on the P2 row B connector, refer to the VMEbus specifications.

Signal(s)	Pin(s)	Description
BCLK4	P2A-1	Network byte clock - 12.5 MHz
~ENDEC	P2A-2	Primary front end encoder/decoder R/W strobe
~ENDEC2	P2A-3	Secondary front end encoder/decoder R/W strobe
~BWR	P2A-4	Buffered write signal; data write line
~AIORDY	P2A-5	29K asynchronous I/O access ready line
~SYNCRST	P2A-6	Synchronous reset signal; generated whenever 29K CPU is reset
~FOR_CMT_INT	P2A-7	Front end FORMAC/CMT interrupt
~PHY1_INT	P2A-8	Not currently used
BPA2 - BPA5	P2A-9 - P2A-12	Buffered processor address bus lines 2 - 5
OBC1	P2A-13	PHY A optical bypass control enable
OBC2	P2A-14	PHY B optical bypass control enable
DATTACH	P2A-15	Dual MAC/single MAC PHY B select
BPD8 - BPD15	P2A-16 - P2A-23	Buffered processor data bus lines 8 - 15
~PHY2_INT	P2A-24	Not currently used
ENDEC1_S0	P2A-25	Status line 0 on the ENDEC of the primary PHY
ENDEC1_S1	P2A-26	Status line 1 on the ENDEC of the primary PHY
ENDEC1_S2	P2A-27	Status line 2 on the ENDEC of the primary PHY
ENDEC2_S0	P2A-28	Status line 0 on the ENDEC of the secondary PHY
ENDEC2_S1	P2A-29	Status line 1 on the ENDEC of the secondary PHY
ENDEC2_S2	P2A-30	Status line 2 on the ENDEC of the secondary PHY
~OBC_PRES	P2A-31	Optical bypass control present; used by software detect when there is an optical bypass switch connected to the 4211
12VN	P2A-32	Power, +12V
XMT0 - XMT9	P2C-1 - P2C-9	FORMAC transmit bus
N/C	P2C-11	No connect
PRI_RCV0 - PRI_RCV9	P2C-12 - P2C-21	FORMAC primary receive bus
NO_CONNECT	P2C-22	No connect
SEC_RCV0 - SEC_RCV9	P2C-23 - P2C-32	FORMAC secondary receive bus

CHAPTER 6
DIAGNOSTICS

OVERVIEW

The 4211 supports a variety of power-up and host-controlled diagnostics. Power-up diagnostics execute automatically each time the board is reset. Host-controlled diagnostics may be invoked individually while the Common Boot interface is running (i.e. before the host has booted the RC Interface using the BOOT command).

If any test fails (power-up *or* host-controlled), the 4211 writes the ASCII value "FAIL" (0x4641494C) to the Command Status Word[13], followed by a set of fields describing the failure.

> **IMPORTANT**
> If a diagnostic test fails, please make note of the returned test data. This will enable Interphase to provide you with better technical assistance and faster repair turnaround.

This chapter describes the controller-specific parts of the 4211's diagnostics. Features which are generic to controllers that support Common Boot are documented in the *Common Boot and RC Interface User's Guide*.

POWER-UP DIAGNOSTICS

Each time the controller is reset or powered-up, it performs a series of power-up diagnostics. These tests, which take about three seconds to complete, check the 4211's memory and internal signal paths.

The tests executed at power-up are the same as the host-controlled diagnostics described later, with the following exceptions:

- In power-up mode, quick memory tests are performed. With host-controlled diagnostics, extended memory tests are done.
- VME DMA tests are not executed in power-up mode, but can be run in host-controlled mode.
- External loopback tests are not executed during power-up diagnostics, but can be invoked by the host after power-up.

At reset, the red LED is on and the green LED is off. The first thing the CPU does is turn the red LED off and the green LED on. This indicates that the CPU is functional and has started executing code. The board then

[13] The Command Status Word is a 32-bit field used solely by the Common Boot interface. On the 4211, this field is located at an offset of +0x80 bytes from the short I/O base address. For more information, see the chapter on Common Boot in the *Common Boot and RC Interface User's Guide*.

performs a buffer RAM test. If the buffer RAM test fails, both the red and green LEDs are turned on and the controller stops. If the RAM test passes, the controller runs all the remaining power-up diagnostics. While the tests are executing, the red and green LEDs toggle on and off successively.

Once the diagnostics finish, the LEDs stop toggling on/off and control is passed to the Common Boot interface. If all the tests complete successfully, the Command Status Word will contain the ASCII value "CBOK" (0x43424F4B), and the green LED will remain on.

If any of the power-up tests fail, the Command Status Word will contain the ASCII value "FAIL" (0x4641494C), and the red LED will remain on. In this event, the Command Status Word will be followed by a series of test-specific fields describing the failure. The returned fields of a failed power-up test typically match those of its host-invoked counterpart. Therefore, to identify which power-up test failed, compare the actual returned data to the returned data structures described in the remainder of this chapter.

HOST-CONTROLLED DIAGNOSTICS

While the Common Boot interface is running, it may be used to perform a variety of operations on the 4211. These include memory peek and poke, LED control, and diagnostic commands. The available commands are documented in the chapter on Common Boot in the *Common Boot and RC Interface User's Guide*.

These tests must be performed before starting up the RC Interface, since Common Boot "goes away" once RC starts up.

The DIAG Command

Most of the commands in the Common Boot command set are generic. Their definition is the same for any controller that supports the Common Boot interface. The DIAG command, however, is an exception. It is used to perform a variety of controller-specific tests. Therefore, the definition of its fields varies depending on the controller ***and*** on the type of test being performed.

Format of DIAG

Figure 6-1 shows the format of the DIAG command for the 4211.

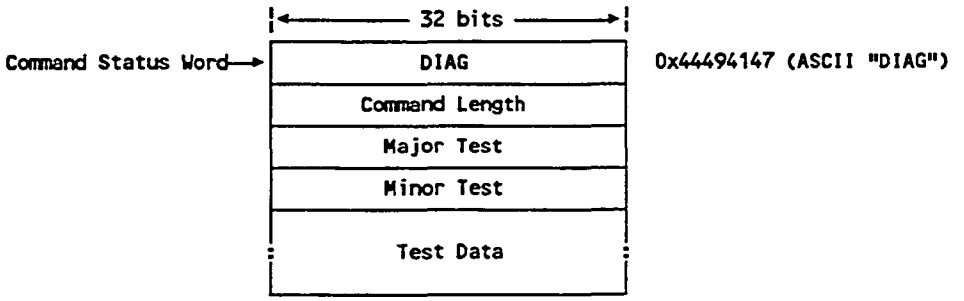


Figure 6-1 . Format of DIAG Command

The meaning of the fields in the DIAG command is as follows:

Command Length. This field contains the total number of bytes in the test command, including itself, but *excluding* the Command Status Word (which contains the ASCII value "DIAG").

Major Test. Each major hardware section of the 4211 has a separate test which is assigned a number (see Table 6-1). A major test number of 0x0 will execute all the major tests.

Minor Test. Some of the major tests are divided into a number of minor tests. A minor test number of 0x0 will execute all the minor tests in the associated major test.

NOTES: [1] If the Major Test field contains 0x0, the Minor Test field is ignored.
[2] If no minor tests are defined for the major test, the host may enter either 0x0 or 0x1 in the minor test field. If the 4211 returns a status block (see Figure 6-2, p. 6-4), the returned Minor Test field will contain the value 0x1.

Test Data. Only VMEbus DMA tests require test-specific data. These fields are provided in the VMEbus DMA test description (p. 6-10).

Table 6–1 . Major Test Numbers

CODE	TEST TYPE
0x0	Execute ALL major tests (1 through 7)
0x1	Memory Test
0x2	Loopback Test
0x3	VME DMA test
0x4	NOVRAM Test
0x5	9513 Counter Test
0x6	CMT PAL Test
0x7	CAM Test

Details on each major test and its minor test(s) are provided in the following subsections. For instructions on how to issue the DIAG command, see the chapter on Common Boot in *Common Boot and RC Interface User's Guide.*

Test Results

If the test completes successfully, the Command Status Word will contain the ASCII value "CBOK" (0x43424F4B). No other data is returned. If the test fails, the 4211 returns the structure shown in Figure 6-2.

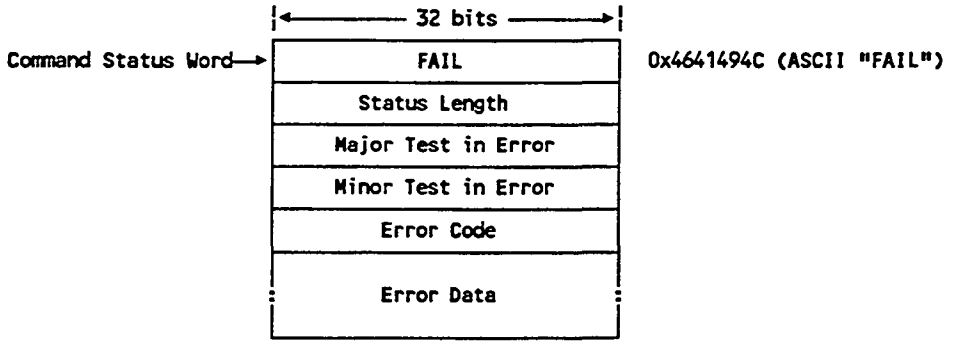


Figure 6–2 . "FAIL" Returned Status Block

The meaning of the fields in the error result structure shown in Figure 6-2 is as follows:

Status Length. This field contains the total number of bytes in the status block, including itself, but *excluding* the Command Status Word (which contains the ASCII "FAIL").

Major Test in Error. This reports the major test number of the failed test.

Minor Test in Error. This reports the minor test number of the failed test, if any. If there is no minor test, this field contains 0x1.

Error Code. Table 6-2 shows a list of possible error codes. Error code 0x1 is for general failure, and can occur in conjunction with any test. Error codes 0x2 — 0xB are associated with individual tests.

Error Data. This field contains test-specific information about the error. Some errors have data associated with them, while others do not. Consult the individual test description for details.

If a given error does *not* include error data, the returned structure will end after the Error Code field. It will consist of four 32-bit fields (Status Length, Major Test in Error, Minor Test in Error, and Error Code).

Table 6–2 . Diagnostic Error Codes

ERROR CODE	ERROR TYPE	TEST(S) WITH WHICH ERROR MAY BE ASSOCIATED
0x1	Test failed	All tests
0x2	Loopback Transmit Timeout	Loopback test
0x3	Loopback Receive Timeout	Loopback test
0x4	VME DMA Timeout	VME DMA test
0x5	VME syserror	VME DMA test
0x6	NOVRAM Checksum Error	NOVRAM test
0x7	Bad 9513 Counter Status	9513 Counter test
0x8	9513 Counter Interrupt Error	9513 Counter test
0x9	CAM did not match on an address	CAM test
0xA	CAM matched on wrong address	CAM test
0xB	Bad CAM full indication	CAM test
0xFF	Bad Major or Minor test code	(User-supplied invalid code)

TEST DESCRIPTIONS

The remainder of this chapter provides a summary of the host-controlled tests, followed by a description of each major test and its minor tests (if any).

Quick Reference Guide

Table 6-3 summarizes the host-controlled diagnostics supported on the 4211. It is a quick reference guide for users who are already familiar with issuing diagnostic commands to the 4211 and interpreting test results.

Table 6–3 . Quick Reference Guide for 4211 Post Power-up Diagnostics

MAJOR TEST	MAJOR TEST TYPE	MINOR TESTS	POSSIBLE ERROR(S)
0x0	Execute ALL major tests (1 through 7)	N/A	If a test fails, the returned status block will identify the major test, minor test (if any), and error code of the failed test.
0x1	Memory Test (p. 6-7)	0x1 Static RAM Test 0x2 Buffer RAM Test 0x3 Short I/O Test	0x1 Test Failed
0x2	Loopback Test (p. 6-8)	0x1 FORMAC Loopback Test 0x2 ENDEC Short Loopback (PHY A) 0x3 ENDEC Loopback (PHY A) 0x4 External Loopback (PHY A) 0x5 ENDEC Short Loopback (PHY B) 0x6 ENDEC Loopback (PHY B) 0x7 External Loopback (PHY B)	0x1 Test Failed 0x2 Loopback Transmit Timeout 0x3 Loopback Receive Timeout
0x3	VME DMA Test (p. 6-10)	0x1 Short I/O VME DMA Test 0x2 Standard VME DMA Test	0x1 Test Failed 0x4 VME DMA Timeout 0x5 VME syserror
0x4	NOVRAM Test (p. 6-12)	N/A	0x1 Test Failed 0x6 NOVRAM Checksum Error
0x5	9513 Counter Test (p. 6-13)	0x1 Test Counter Chip 1 0x2 Test Counter Chip 2	0x1 Test Failed 0x7 Bad 9513 Counter Status 0x8 9513 Counter Interrupt Error
0x6	CMT PAL Test (p. 6-15)	0x1 Test CMT PAL (PHY A) 0x2 Test CMT PAL (PHY B)	0x1 Test Failed
0x7	CAM Test (p. 6-16)	N/A	0x1 Test Failed 0x9 Frame Match Error 0xA Unmatched Frame Error 0xB CAM "Full/Not Full" Error

NOTE: The numbers in parenthesis indicate the page in this chapter on which the major test description can be found.

Major Test 0x1 : Memory Test

This test performs an extended (Walking 1s) test on the 4211's memory. Memory is first written and then read to check for correctness. (NOTE: During power-up diagnostics, only a quick memory test is done.)

Table 6-4 lists the minor tests which are available.

Table 6-4 . Minor Tests of Memory Test (0x1)

CODE	NAME	FUNCTION
0x1	Static RAM test	Test static RAM on the board.
0x2	Buffer RAM test	Test buffer/video RAM on the board.
0x3	Short I/O test	Test shared memory on the board.

If the test completes successfully, the 4211 writes CBOK to the Command Status Word. If it fails, FAIL is written to the Command Status Word.

Possible Errors

Only one error code is associated with the memory test — 0x1 (Test Failed). This error indicates that the value read from memory is not same as what was written to it. The returned structure is shown in Figure 6-3.

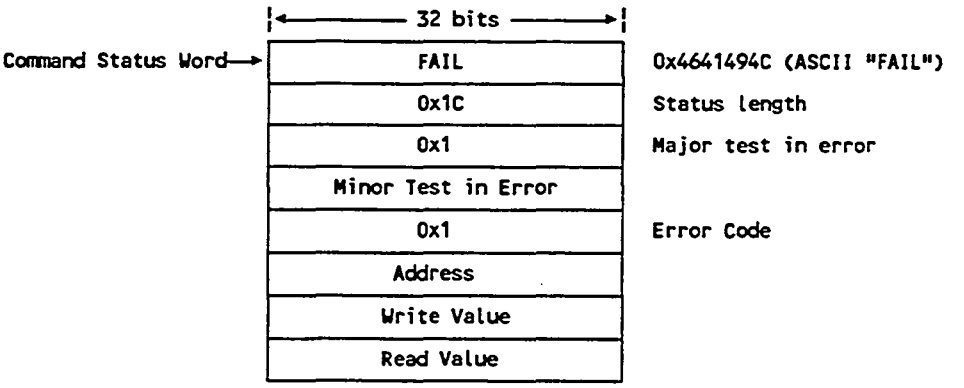


Figure 6-3 . Returned Data for Failed Memory Test (Error Code 0x1)

The following fields of error data are returned:

- Address - Board address where memory test failed.
- Write Value - Value that was written to the failed location.
- Read Value - Value that was read from the failed location.

Major Test 0x2 : Loopback Test

This test checks the SUPERNET chipset. An FDDI frame is built in buffer RAM and is looped back though the SUPERNET chipset. The received frame is then compared with the transmitted frame and verified for correctness.

The test performs the following actions:

1. Set promiscuous mode (receive everything)
2. Build test frame
3. Send test frame
4. Receive test frame

The test frame consists of:

Frame Control (FC)	1 byte	set to 0x40
Source Address (SA)	6 bytes	set to 0x55 0x55 0x55 0x55 0x55 0x55
Destination Address	6 bytes	set to <actual board address>
Data	51 bytes	set to <data pattern>
Total frame length:	**64 bytes**	

Table 6-5 lists the minor tests which are available.

Table 6–5 . Minor Tests of Loopback Test (0x2)

CODE	NAME	FUNCTION
0x1	FORMAC loopback test	Frame is looped through FORMAC chip
0x2	ENDEC short loopback (PHY A)	Frame is looped through ENDEC chip (PHY A)
0x3	ENDEC loopback (PHY A)	Frame is looped through ENDEC and EDS chips (PHY A)
0x4	External loopback (PHY A)	Frame is looped through fiber.
0x5	ENDEC short loopback (PHY B)	ENDEC short loopback (PHY B)
0x6	ENDEC loopback (PHY B)	Frame is looped through ENDEC and EDS chips (PHY B)
0x7	External loopback (PHY B)	Frame is looped through fiber.

NOTE: The external loopback tests (0x4 and 0x7) are not executed during power-up diagnostics. These tests require a loopback fiber cable between the optic transmitter and receiver.

If the test completes successfully, the 4211 writes CBOK to the Command Status Word. If it fails, FAIL is written to the Command Status Word.

Possible Errors

Three error codes are associated with loopback tests:

— 0x1 (Test Failed) The transmitted frame did not match the received frame.
— 0x2 (Loopback Transmit Timeout) The front end failed to post a transmit done interrupt
— 0x3 (Loopback Receive Timeout) The front end failed to post a receive done interrupt.

Error codes 0x2 and 0x3 do not return error data (see discussion of Error Data field on p. 6-4). Error 0x1 does (see Figure 6-4).

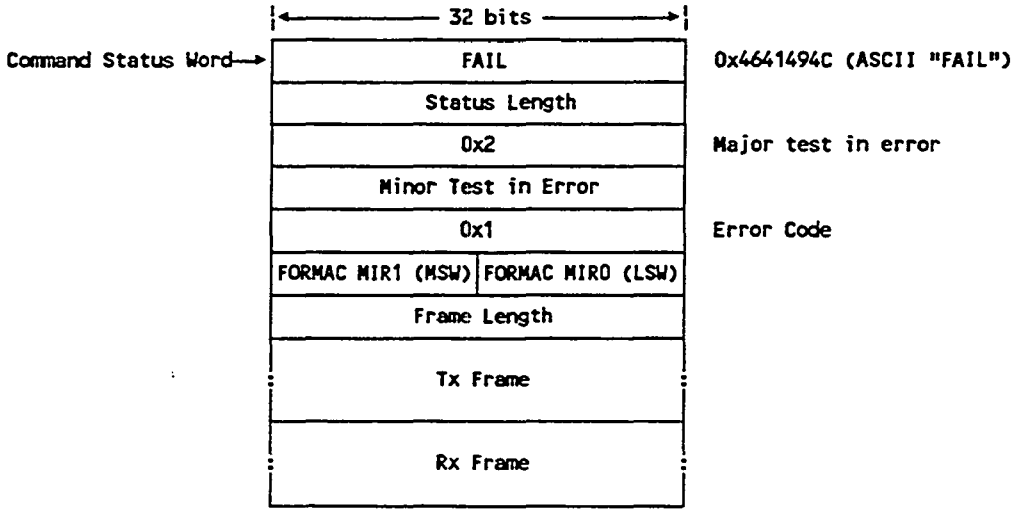


Figure 6–4 . Returned Data for Failed Loopback Test (Error Code 0x1)

The following fields of error data are returned:

- FORMAC MIR1 / FORMAC MIR0 - Value in FORMAC MIR 1 and FORMAC MIR 0 register.
- Frame Length - Length of the loopback frame (a maximum of 64 bytes, including header).
- Tx Frame - Transmitted frame of length 'Frame Length'.
- Rx Frame - Received frame of length 'Frame Length'.

Major Test 0x3 : VMEbus DMA Test

This test checks the 4211's VME interface. Data is first written to the address specified by the DMA command, and then read back. The data that is read back is compared for correctness. The VMEbus DMA test is not executed during power-up diagnostics.

The VMEbus DMA test requires a special form of the DIAG command, as shown in Figure 6-5.

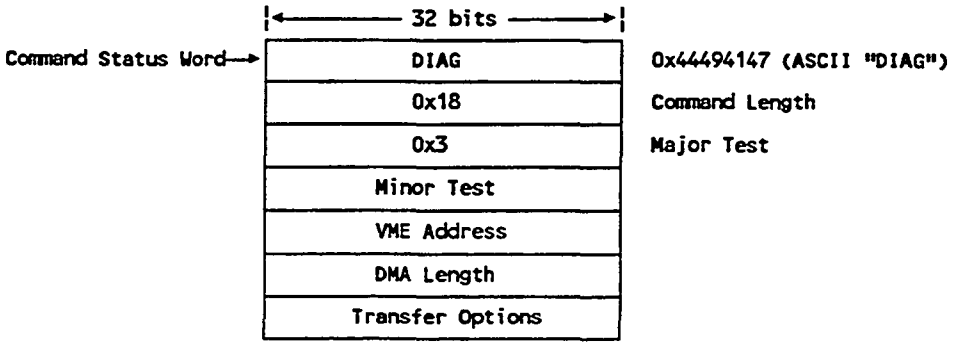


Figure 6-5 . Format of DIAG Command for VMEbus DMA Command

- Minor Tests - Table 6-6 lists the minor tests which are available.
- VME Address - Start of VME memory where data is DMA'ed.
- DMA Length - Length of DMA in longwords.
- Transfer Options - This is the Transfer Options Word as defined on p. 4-4.

Table 6-6 . Minor Tests of VMEbus DMA Test (0x3)

CODE	NAME	FUNCTION
0x1	Short I/O VME DMA Test	Tests board's slave DMA circuitry.
0x2	Master VME DMA Test	Tests the board's master DMA circuitry.
0x4	Raw VME DMA Test	Bus read/write, no data generation/compare

NOTES: [1] When running test 0x1 (Short I/O VME DMA Test), make sure that the address specified in the VME Address field is in the 4211's short I/O memory. Also ensure that the DMA Length field contains a value which limits DMA transfers to the ***short I/O region only***. In other words, do not specify values which cause the 4211 to perform DMA transfers outside its 512-byte short I/O space.

[2] When running test 0x2 (Master VME DMA Test), the address specified in the VME Address field should be a VME memory address to which the board can DMA test data.

[3] If desired, the host may specify a DMA address in the command area itself. This causes the command fields to be overwritten while the test is running, but otherwise does not affect board operations. Once the test is complete, CBOK or FAIL will be written to the Command Status Word, followed by additional data if the test failed (Figure 6-6).

[4] When running test 0x4 (Raw VME DMA Test), the user builds the desired data pattern. The controller reads the VMEbus and then writes the same data back to memory.

If the test completes successfully, the 4211 writes CBOK to the Command Status Word. If it fails, FAIL is written to the Command Status Word.

Possible Errors

Three error codes are associated with the VMEbus DMA tests:

— 0x1 (Test Failed)
— 0x4 (VME DMA Timeout) The board failed to post a DMA done interrupt.
— 0x5 (VME syserror) A VME syserror occurred during VME DMA.

Error codes 0x4 and 0x5 do not return error data (see discussion of Error Data field on p. 6-4). Error 0x1 does (see Figure 6-6).

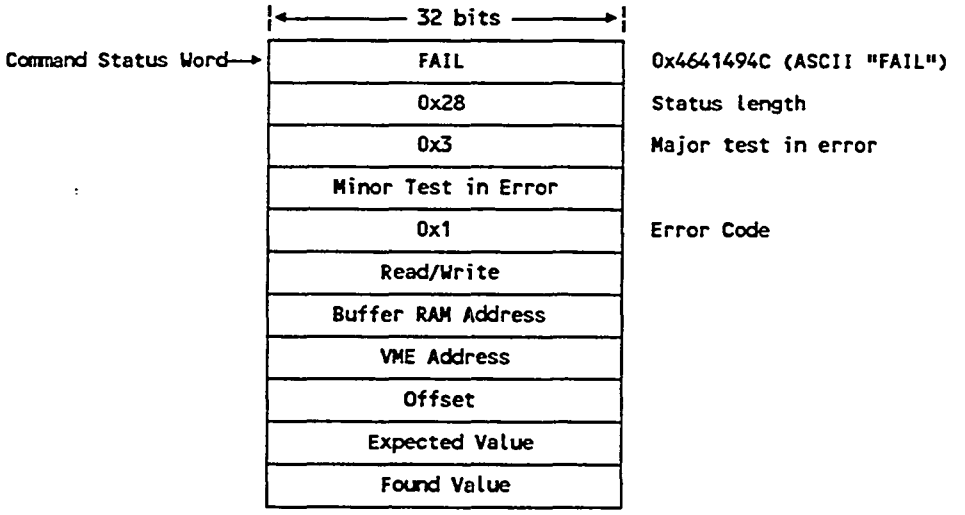


Figure 6–6 . Returned Data for Failed VME DMA Test (Error Code 0x1)

The following fields of error data are returned:

- Read/Write - 0=Read, 1=Write. Specified whether the error occurred during DMA write or read. Only valid for short I/O test.
- Buffer RAM address - Starting buffer RAM address for DMA.
- VME address - Starting VME address for DMA.
- Offset - Longword offset from start where error occurred.
- Expected Value - Value that was DMA'ed.
- Found Value - Value found in memory.

Major Test 0x4 : NOVRAM Test

This test checks the 4211's NOVRAM (X2444 - Xicor Nonvolatile Static RAM). The test is done by writing to all bits of static RAM, reading them back, and comparing them. NOVRAM has 256 memory bits which are logically divided into 16-bit words for this test.

Note that this test only writes to the temporary storage area of the NOVRAM. The permanent storage area, which is E2PROM, is *not* written to by the test. This is due to a physical constraint, namely that data can be written to E2PROM only a limited number of times. Therefore, the diagnostics simply verify the checksum in permanent storage where MAC addresses are stored.

There is only one NOVRAM test. No minor tests are defined.

If the test completes successfully, the 4211 writes CBOK to the Command Status Word. If it fails, FAIL is written to the Command Status Word.

Possible Errors

Two error codes are associated with the NOVRAM test:

— 0x1 (Test Failed) The word read from NOVRAM is not the same as the word written to it.
— 0x6 (NOVRAM checksum error) The NOVRAM has a bad checksum.

Error code 0x6 does not return error data (see discussion of Error Data field on p. 6-4). Error 0x1 does (see Figure 6-7).

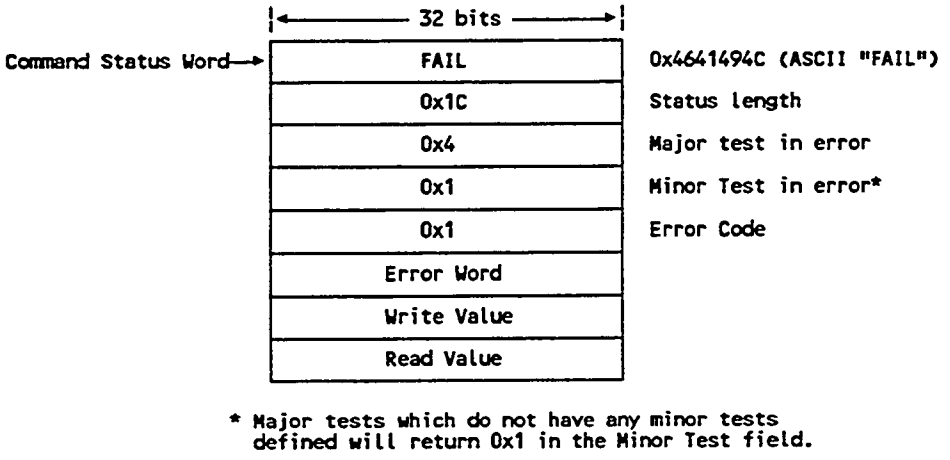


Figure 6–7 . Returned Data for Failed NOVRAM Test (Error Code 0x1)

The following fields of error data are returned:

- Error Word - Word in error
- Read Value - Value read back from that word
- Write Value - Value written in that word

Major Test 0x5 : 9513 Counter Test

This test checks the 4211's Am9513 counters. Counters are tested in two steps. In the first step, counter registers are written, read back, and verified for correctness. The second test verifies that the counters generate interrupts when the terminal count is reached in an armed counter.

Table 6-7 lists the minor tests which are available.

Table 6–7 . Minor tests of 9513 Counter Test (0x5)

CODE	FUNCTION
0x1	Test counter chip 1
0x2	Test counter chip 2

If the test completes successfully, the 4211 writes CBOK to the Command Status Word. If it fails, FAIL is written to the Command Status Word.

Possible Errors

Three error codes are associated with the counter tests:

- 0x1 (Test Failed) A value read from a counter register does not match the value written to it.
- 0x7 (Bad 9513 Counter Status) A status register read from the counter chip is not correct.
- 0x8 (9513 Counter Interrupt Error) The counter chip either: 1) generated an interrupt when it should not have, or 2) failed to generate an interrupt when it should have.

All three of the above error codes return error data.

If error code 0x1 (Test Failed) occurs, the 4211 returns the structure shown in Figure 6-8.

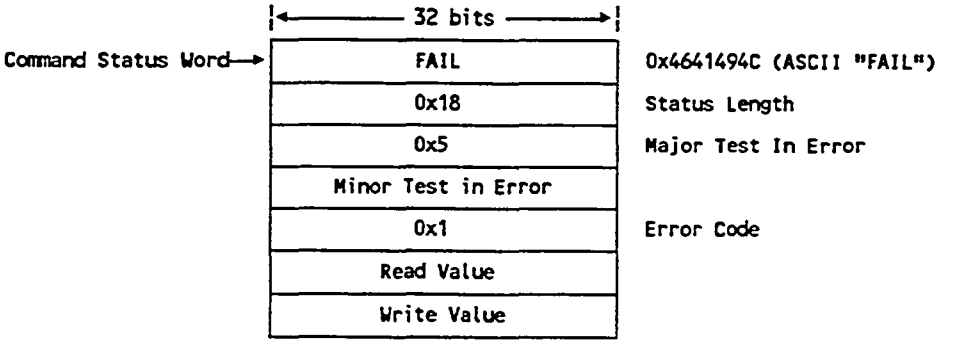


Figure 6–8 . Returned Data for Failed 9513 Counter Test (Error Code 0x1)

The following fields of error data are returned:

- Write value - Value written to a counter register
- Read value - Value read from that counter register.

If error code 0x7 or 0x8 occur, the 4211 returns the structure shown in Figure 6-9.

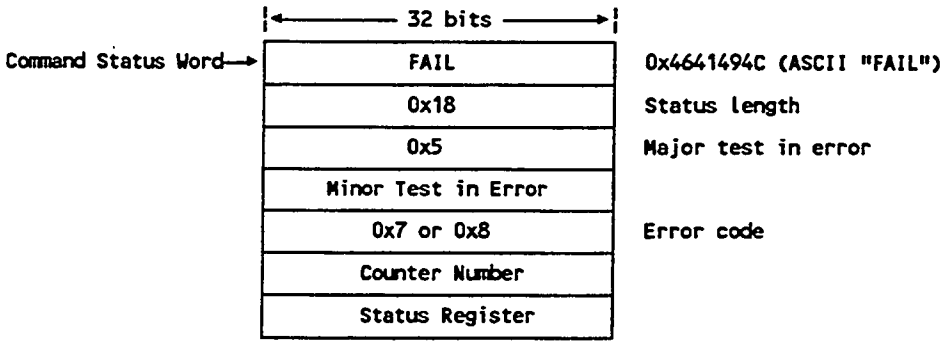


Figure 6–9 . Returned Data for Failed 9513 Counter Test (Error Codes 0x7 or 0x8)

The following fields of error data are returned:

- Counter Number - Counter number that generated error
- Status register - Value in status register when error occurred

Major Test 0x6 : CMT PAL Test

This test checks the CMT PALs by: 1) putting the ENDEC associated with the PAL in loopback mode, 2) forcing different line states, and 3) verifying that the PAL detects each forced line state change.

Table 6-8 lists the minor tests which are available.

Table 6–8 . Minor Tests of CMT PAL Test (0x6)

CODE	NAME
0x1	Test CMT PAL (PHY A)
0x2	Test CMT PAL (PHY B)

If the test completes successfully, the 4211 writes CBOK to the Command Status Word. If it fails, FAIL is written to the Command Status Word.

Possible Errors

Only one error code is associated with the CMT PAL test — 0x1 (Test Failed). This error occurs if the CMT PAL generates an interrupt even though the line state did not change from one state to another. It also occurs if the CMT PAL fails to generate an interrupt when the line state changes from one forced state to another.

The returned structure is shown in Figure 6-10.

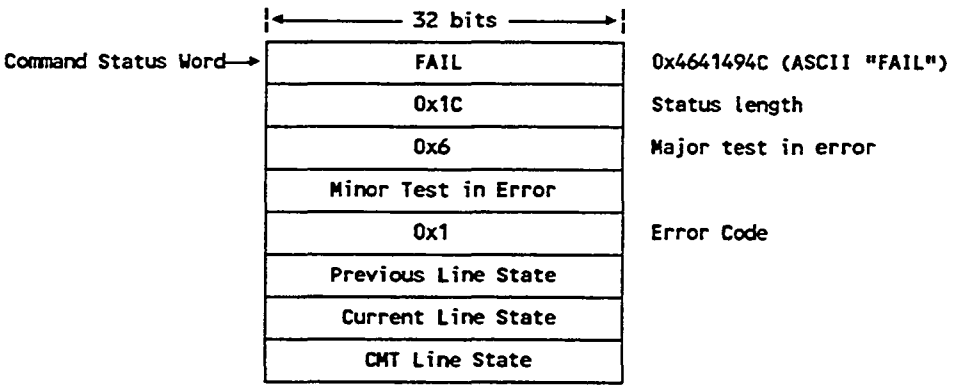


Figure 6–10 . Returned Data for Failed CMT PAL Test (Error Code 0x1)

The following fields of error data are returned:

- Previous line state - Previous forced ENDEC line state.
- Current line state - Current ENDEC line state.
- CMT line state - Line state read from CMT PAL when error occurred.

Major Test 0x7 : CAM Test

This routine tests the 4211's CAM (Am99C10 - 256 x 48 Content Addressable Memory). The CAM is tested in two steps. First, all 256 48-bit CAM registers are written, read back, and compared. Second, the test loops back two frames. One of the frames has a destination address which is stored in the CAM; the other has a destination address which is *not* in the CAM. The test verifies that the first frame is received, while the second is not.

There is only one CAM test; therefore, no minor tests are defined.

If the test completes successfully, the 4211 writes CBOK to the Command Status Word. If it fails, FAIL is written to the Command Status Word.

Possible Errors

Four error codes are associated with the CAM test:

- 0x1 (Test Failed) The address written to a CAM does not match the address read from a CAM register.
- 0x9 A frame was not received, even though its destination address is present in CAM.
- 0xA A frame was received, even though its destination address was not present in CAM.
- 0xB The CAM indicates that it is full before all 256 entries are put it, or it indicates that it is not full after all 256 entries are put in it.

Error codes 0x9, 0xA, and 0xB do not return error data (see discussion of Error Data field on p. 6-4). Error 0x1 does (see Figure 6-11).

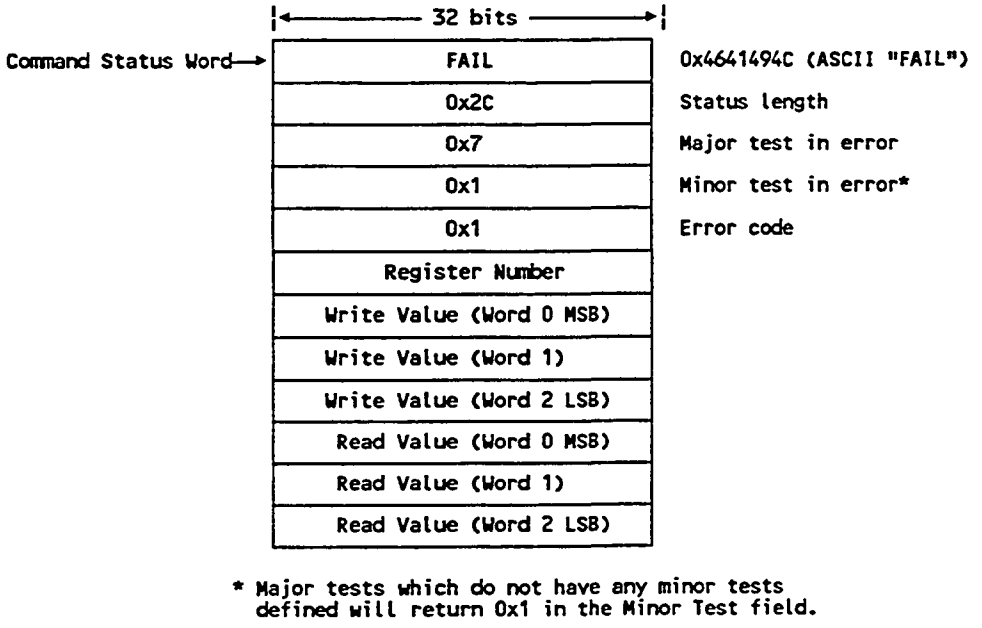


Figure 6–11 . Returned Data for Failed CAM Test (Error Code 0x1)

The following fields of error data are returned:

- Register Number - CAM register that failed
- Write value - Value written to CAM register (three 16-bit words)
- Read value - Value read back from CAM register (three 16-bit words)

This page is intentionally left blank.

APPENDIX A
FDDI TUTORIALS

OVERVIEW

This appendix contains reprints of the following articles:

[1] "FDDI — A Perspective," by Floyd E. Ross. This article is reprinted from pp. 170 - 186 of *The Local Area Network Sourcebook, 5th Edition* with permission of Phillips Publishing Inc., 7811 Montrose Road, Potomac, MD, 20854, (301)340-2100.

Floyd E. Ross is senior staff engineer with Unisys, located in Wayne, Pennsylvania, with responsibilities in LAN definition and implementation. Educated in Canada, he was granted a B.A. in Mathematics and Physics from Mount Allison University in 1955 and did postgraduate study at McMaster University. He is an active contributor to accredited standards committee ASC X3T9 on I/O interfaces and to IEEE P802. Mr. Ross is vice chairman of ASC X3T9.5 and has been the leader of the FDDI standards development effort since its inception in that committee. He has presented and authored a number of papers on local area networks and FDDI.

[2] "Fiber Distributed Data Interface (FDDI) — A Tutorial," by Mark S. Wolter. This article is reprinted from pp. 16 - 26 of *ConneXions — The Interoperability Report*, Vol 4., No. 10. October, 1990 with permission of Interop, Inc., 480 San Antonio Road, Suite 100, Mountain View, CA, 94040, (415)941-3399.

Mark Wolter has worked at National Semiconductor for over six years, and is an application manager for FDDI products. Prior to his work with FDDI, he worked on the design and implementation of disk data and networking controller chips. He played a leading role in the development of a design methodology incorporating the use of CAE tools and standard sells in the development, simulation, verification, and system modelling of complex VLSI chips. He holds a B.S. in Electrical Engineering, with a concentration in Computer Engineering, from Cornell University.

"FDDI — A Perspective" begins on p. A-3. "Fiber Distributed Data Interface (FDDI) — A Tutorial" starts on p. A-21.

This page intentionally left blank.

FDDI — A PERSPECTIVE

by Floyd E. Ross

This article is reprinted from pp. 170 - 186 of *The Local Area Network Sourcebook, 5th Edition.* Reprinted with permission of Phillips Publishing Inc., 7811 Montrose Road, Potomac, MD, 20854, (301)340-2100.

The fiber distributed data interface (FDDI) has achieved wide recognition as the high-speed local area network (LAN) of the next decade. FDDI uses a token ring architecture and employs optical fiber as the transmission medium. This article explores the basis for that recognition, provides an overview of FDDI's functionality, updates FDDI's current status, and provides a perspective on this high-speed LAN. Both the basic FDDI – which offers packet service – and FDDI-II – which adds a circuit-switched service to create an integrated services LAN – are addressed.

BACKGROUND

FDDI is being developed in Accredited Standards Committee (ASC) X3T9 – chartered to develop computer input/output (I/O) interface standards. In the late 1970s, ASC X3T9 recognized the need for a new I/O channel standard as an alternative to federal information processing standards (FIPS) 60-63. In late 1982, work was just starting on FDDI in ASC X3T9.

Meanwhile, the four-Mbps token ring (802.5) protocol definition was emerging from the Institute of Electronic and Electrical Engineers (IEEE) P802 standards project. IEEE P802 was formed in 1980 to develop LAN standards with data rates up to 20 Mbps (LAN standards with rates up to 50 Mbps and above were made the purview of ASC X3T9). This token ring architecture formed the basis of the FDDI protocol intended for use at the much higher rate of 100 Mbps. In this same time frame, increased emphasis was being placed on high-performance, graphic workstations. These would require a LAN of greater performance than was offered by the IEEE P802 LANs.

Another factor that strongly influenced the evolution of FDDI was the development of a new generation of digital PBXs in the early 1980s. These offered a new level of service, including both packet and circuit switching data transfers, but were dependent upon the development of a high-performance backbone LAN with both packet and circuit switching capabilities. FDDI was viewed as having the potential for enhancement (FDDI-II) to satisfy these diverse needs.

Meanwhile, in the standards arena, the open systems interconnection (OSI) model had been put in place. This layered the design of computer interconnections, and allowed the development of separate standards for the different layers. It also provided a proper framework for the development of a set of FDDI standards.

Floyd E. Ross is senior staff engineer with Unisys, located in Wayne, Pennsylvania, with responsibilities in LAN definition and implementation. Educated in Canada, he was granted a B.A. in Mathematics and Physics from Mount Allison University in 1955 and did postgraduate study at McMaster University. He is an active contributor to accredited standards committee ASC X3T9 on I/O interfaces and to IEEE P802. Mr. Ross is vice chairman of ASC X3T9.5 and has been the leader of the FDDI standards development effort since its inception in that committee. He has presented and authored a number of papers on local area networks and FDDI.

FDDI AS THE SOLUTION

FDDI has won acceptance by a diversified community of computer and communications applications. Some interesting factors are responsible for this acceptance. The IEEE P802 effort provided several medium-speed (1 to 20 Mbps) LANs. In effect, IEEE P802 has popularized the LAN. Popularizing the LAN has created a market for a higher-speed LAN to perform the backbone function for lower-speed IEEE 802 LANs and also to satisfy applications that require a higher-performance LAN.

Another key factor to the acceptance of FDDI is the dramatic improvement in the price and performance characteristics of optical fiber and related components such as optical transmitters and receivers. When viewed in the light of the many other advantages that the use of optical fiber offers - high data bandwidth, security, safety, immunity to electromagnetic interference, and reduced weight and size - these improved price and performance characteristics make the concept of an all-fiber LAN most attractive. The FDDI design has been optimized to the use of optical fiber since its inception. As a result, FDDI is now in a leadership role in the development of optical fiber LAN technology.

Finally, the value of standardization has been maximized for FDDI. This is the age when the true value of standards is being recognized in all marketplaces - from the individual customer to the federal government. With the emphasis on standards, FDDI has conformed to the ISO Model and has integrated sufficient functionality and satisfied a number of market segments to such a degree that FDDI can be the one standard satisfying all the requirements of the broad, high-speed LAN marketplace.

Put into perspective, the FDDI standard may not be the optimum solution for all LAN applications. However, it is an adequate one. The value of one high-speed optical fiber LAN standard is immense. As a result, a high level of cooperation has been realized between the many supporters of FDDI. This cooperation has led to FDDI's functional richness and success.

ADVANTAGES OF A RING CHOICE FOR FDDI

Reliability

The importance of reliability (including availability and serviceability) and survivability, despite physical damage to a network, cannot be overstressed. With proper design, including suitable physical partitioning of the hardware implementation, a counter-rotating trunk ring design with a dual attachment capability - that is, stations with two physical protocol (PHY) entities - offers a superior design choice from a reliability perspective.

Hardware Simplicity

The simplicity of the point-to-point connections of a ring offers a number of advantages. Connections to such a network are simplified from a hardware and a specification perspective. The latter is an important factor in developing a standard intended for interconnecting equipment from a variety of manufacturers.

The point-to-point connections of a ring provide another advantage in that different media can be used for those connections of the ring where unique requirements, either environmental or distance-related, are identified.

Optical Fiber

By far the biggest advantage of the point-to-point connection characteristic of a ring is that it so readily accommodates the use of optical fiber. Because of the high bandwidth offered by optical fiber, the use of bit-serial transmission protocol has significantly reduced the size, cost, and complexity of the hardware required by a network. In I/O channel applications, for example, FDDI realizes the data throughput of eight 48-pin coaxial cable connectors with one duplex optical connector.

Ease of Configuration and Reconfiguration

A ring offers some significant advantages in the ease of initial configuration and reconfiguration as the needs of a network change. Failing stations or links can be readily isolated through the use of appropriate protocols. These same

techniques can be used to enable the logical addition and deletion of stations without detrimental effects on existing ring traffic.

The actual physical addition or removal of stations from a network is relatively easy since each link is an independent connection. Automatic failure isolation, recovery, initialization, and reconfiguration mechanisms ensure minimal impact upon existing ring traffic, even when the cables are being rearranged.

A ring inherently imposes no restrictive logical limit on the length of ring links, the number of stations or the total extent of the network that can be accommodated.

Ring Performance

The figure of merit for network performance may be simply stated as the product of the maximum size, maximum number of stations and the peak data rate for the network under consideration. Let us call this the P-factor. When the P-factor desired exceeds a certain value, a ring design offers significant advantages when compared to other topology choices, such as a bus.

For FDDI, the P-factor is 5×10^{12}, where the units are station-kilometer-bits-per-second. Current information indicates that this can feasibly be extended upwards by a multiplicative factor of at least 10.

Consider the P-factors of other networks. For an IEEE 802.3 (Ethernet) network – a linear bus topology – the P-factor may approach 10^{10}. Such a large-sized network would show poor performance characteristics, even under moderate impressed loads. An upper limit on the order of 10^8 to 10^9 for P-factor seems more realistic for IEEE 802.3 networks.

For IEEE 802.4 – a linear bus topology using a token access method – the P-factor may approach 10^{11}, but again, such a network would exhibit extremely poor performance characteristics. On a practical level, something less than 10^{10} is a more realistic upper limit for the P-factor of IEEE 802.4 networks.

The upper limit for the P-factor of the present I/O Channel (FIPS 60), which is a nine-bit parallel bus design, is approximately 10^8.

Network applications requiring P-factors in the range of ^{9} to 10^{14} station-kilometer-bits-per-second are best implemented with ring topologies. The reasons behind this conclusion are numerous. Network utilizations far exceeding 90 percent, and even 98 percent, are readily achievable, even for large networks with loads exceeding full capacity, using a ring topology.

Rings are insensitive to load distribution and the performance achieved is not degraded significantly by the presence of inactive stations. Token rings provide time-bounded access delay, even under impressed loads approaching the full capacity of the ring.

If counter-rotating rings are used with stations each having two media access controls (MACs), then the secondary ring may also be used for data. This effectively doubles the data handling capacity of the ring when no failures are present.

Continuously-available clock signals reduce the overhead of synchronizing preamble, that results in a significance performance benefit, particularly with short frames.

The fair allocation of bandwidth between competing stations, as well as the guaranteeing of the required bandwidth among the stations of a ring, are easily accomplished. Most importantly, the required protocols can be integrated as part of the token capture process, allowing complex priority algorithms to be implemented with little performance penalty.

Rings offer a real advantage in the low arbitration time required to access the medium. The maximum arbitration time for a ring is equal to the ring latency (the time for a token to propagate around the ring under no load conditions), with the potential for the average to be considerably less. In the case of FDDI, which has multiple-frame transmission capability, the arbitration time

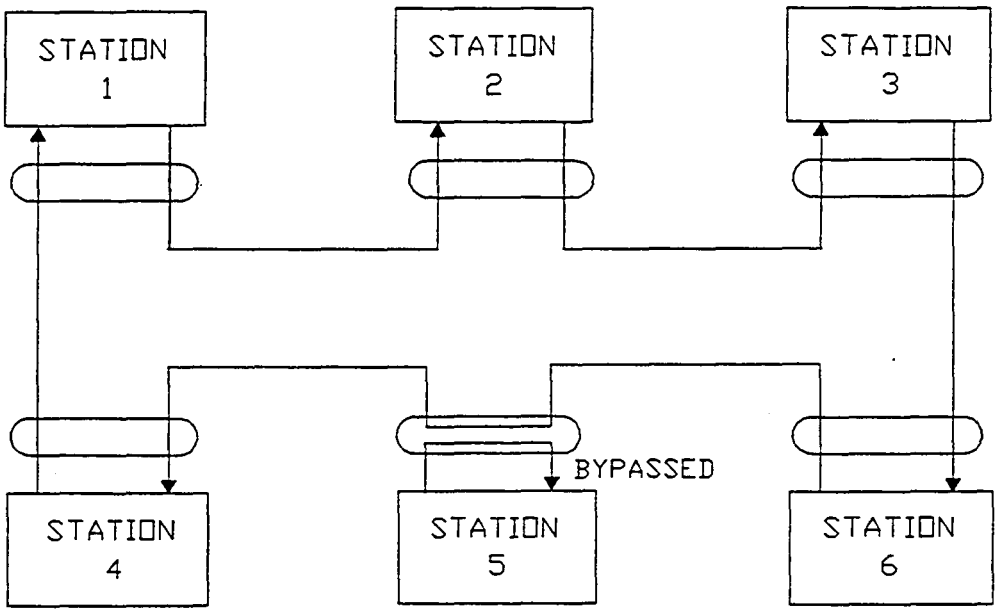


Figure A-1.
BYPASS CAPABILITY

decreases below the time required to pass a token between adjacent stations as the impressed load is increased toward infinity.

Full-Duplex Capability

Yet another advantage that a ring offers is that a ring design, offering two point-to-point connections between user stations, readily offers a full-duplex connection. A single 100-Mbps FDDI ring can carry something approaching 200 Mbps of circuit switched data in full-duplex applications.

WHY NOT A RING SOONER?

If a ring can cost-effectively yield P-factors in the range of 10^9 to 10^{14} and offer other advantages as well, why are rings not in more common usage? The easy answer is that the need for such connectivity and peak data rates was not apparent in the marketplace until recently. But that is an oversimplification. There are complexities unique to a ring that have, until recently, delayed widespread application.

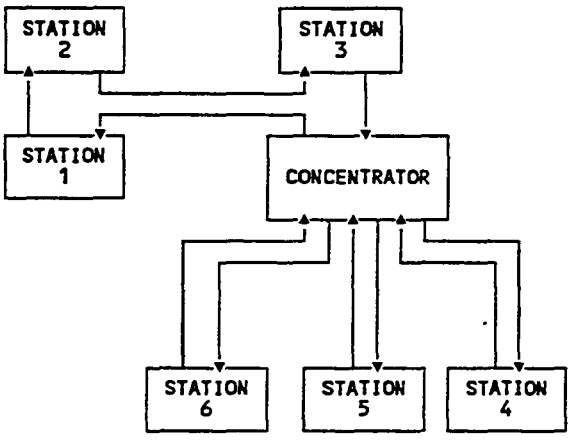


Figure A-2.
CONCENTRATOR CONCEPT

As a first step toward complexity, counter-rotating rings may be desired. Next, it must be recognized that each station is in the path of communications between other stations on the ring, and that provisions must be made in each station to operate on the ring in a consistent and reliable manner as part of the total ring network. Such provisions can be made, but do add complexity to each of the stations. This new level of complexity requires tools, in both the architectural and hardware sense, before a ring can become viable. The open system interconnection (OSI) model and the work of IEEE 802 satisfied most of the architectural needs. The demand for increased hardware complexity is satisfied by the same VLSI semiconductor technology that has made processors so small and fast – a fact that created the market for higher-performance networks in the first place.

Looked at in another way, a ring offers a large advantage in the data rates, extent and connectivity that can be realized. A ring station, however, is more complex and was not viable in the marketplace until high data rates and connectivities were needed in an application's sense – justifying the effort required – and the technology was in place to make rings a cost-effective solution.

FDDI CHARACTERISTICS

FDDI uses optical fiber with light emitting diodes (LEDs) transmitting at a nominal wavelength of 1325 nanometers. Connections between stations are made with a dual fiber cable employing a polarized duplex connector.

The data transmission rate is 100 Mbps. The effective sustained data rate at the data link layer can be well over 95 percent of this peak rate. The four out of five code used on the optical fiber medium requires a 125 megabaud transmission rate. The nature of the clocking limits maximum frame size to 4500 octets. Multiple frames may, however, be transmitted on the same access opportunity.

A total of 1000 physical connections and a fiber path of 200 kilometers has been used as the basis for calculation of the default values of the recovery timers. Considering reconfiguration requirements, these choices allow a maximum configuration of 500 stations (each station represents two physical connections) linked by 100 kilometers of duplex cable. There is no minimum configuration requirement.

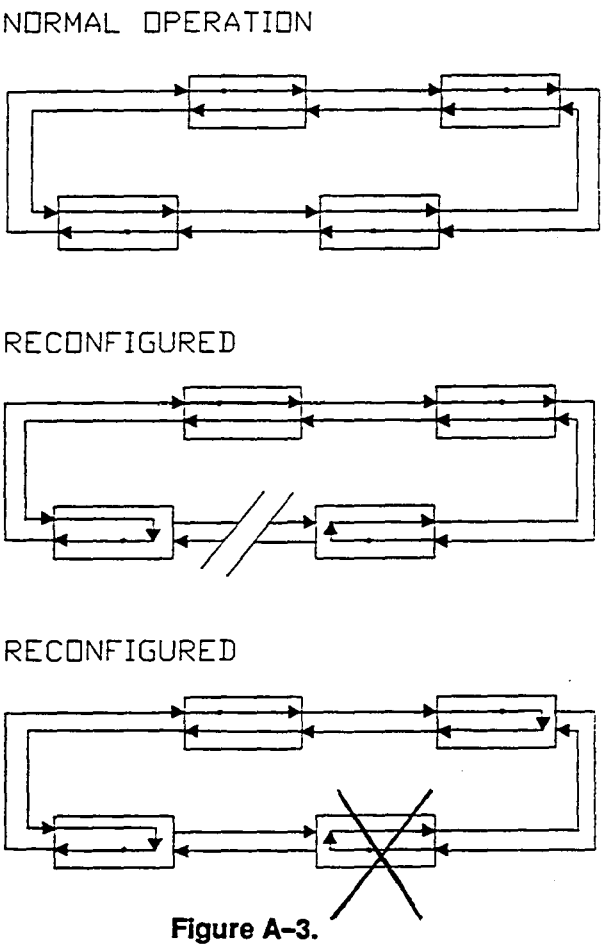


Figure A-3.
COUNTER-ROTATING CONCEPT

Reliability Provisions

A key advantage of ring topologies is that a ring in one topology for which it can be guaranteed that a failing attachment can be isolated. Stations may offer a bypass capability as shown in Figure A-1 that shows station #5 in the bypassed state. A switch – electrical or optical – bypasses a station's receiver and transmitter connections so that the signal from the previous station is passed directly to the next station. Bypassing may be activated by a station itself, at the instigation of its neighbors, by a human operator, automatically at the removal of power, or by some overall network-controlling function.

Yet another approach is the use of concentrators, as shown in Figure A-2, that are attached directly to the trunk ring and in turn provide drop connections for a number of stations – in this case stations #4, 5 and 6. A concentrator may then monitor all its slave stations and isolate any faulty station. It may also provide for the graceful insertion and removal of stations from the ring.

An added level of fault tolerance can be provided by the concept of counter-rotating rings, as shown in Figure A-3. Counter-rotating rings are part of the basic FDDI structure. The counter-rotating ring concept uses two rings connected to each station or concentrator – one clockwise and the other counter-clockwise. In effect, a redundant path is provided to that the ring may stay functional in the event of any single failure. When a failure in the link occurs, the stations on either side reconfigure internally as shown in the middle diagram of Figure A-3. The functional stations make use of the connection in the reverse direction to close the ring, eliminating the bad link. In this figure, the dark dots represent a MAC attachment within the stations. Should a station itself fail, as shown in the bottom diagram of this figure, then the stations on either side would reconfigure to eliminate the failing station and both of the links to it.

The use of all three techniques allows FDDI networks to be configured to tolerate a variety of station or link failures and physical network configurations without any catastrophic consequences. When any of these occur, the network automatically reconfigures, thus eliminating any failing element and maintaining ring operation. Continuous monitoring of the failed link, or station, allows the network to automatically reconfigure and restore normal operation when repair is effected. Any of these reconfigurations may result in the loss of individual frames.

Data Encoding

Information on the medium is in a four or five group code with, with each code group being called a symbol. Of the 32-member symbol set, 16 are data symbols, each representing four bits of ordered binary data. Three are used for starting and ending delimiters, two are used as control indicators and three are used for line-state signaling which is recognized by the physical layer hardware. The remaining eight symbols of the symbol set are not used since they violate code run length and DC balance requirements. Note that QUIET, the symbol that represents no line activity, is a necessary member of the line-state symbol set since it is used to detect the absence of a functioning link.

Frame and Token Formats

Information is transmitted on the FDDI ring in frames. Tokens are used to signify the right to transmit data. Figure A-3 shows the fields that are used within the frame and token.

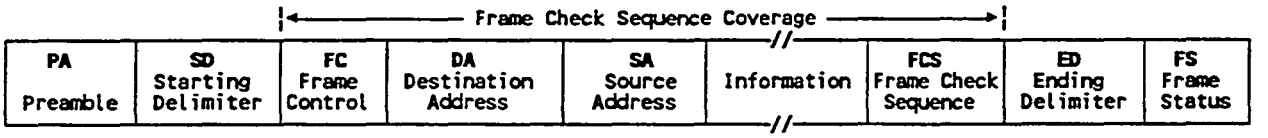
The preamble (PA) field, consisting of IDLE line-state symbols (a maximum frequency signal which is used for establishing and maintaining clock synchronization), precedes every transmission. The starting delimiter (SD) field consists of a two symbol sequence (JK) that is uniquely recognizable, independent of previously established symbol boundaries. The SD establishes the symbol boundaries for the content that follows.

The frame control (FC) field defines the type of frame and its characteristics. It distinguishes between synchronous and asynchronous frames, the length of the address fields (16 or 48 bits), and the kind of frame (e.g., LLC or SMT). One set of FC values is reserved for implementer frames that have no defined format and are to be repeated unchanged by all conforming FDDI stations. The contents of the FC field also uniquely distinguish tokens. Two ending delimiter (ED) symbols (TT) complete a token. Two kinds of tokens, distinguished by their FC field, are defined. These are referred to as restricted and nonrestricted tokens.

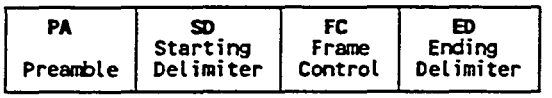
The destination address (DA) and source address (SA) fields are the destination and source addresses. Both may be either 16 or 48 bits long depending on the FC value. DA may be either an individual or a group address, the latter of which has the potential to be recognized by more than one station.

The 32-bit frame check sequence (FCS) field is a cyclic redundancy check using the ANSI standard polynomial. The information field of a frame,

FRAME FORMAT:



TOKEN FORMAT:



PA = Preamble (16 or more symbols)
SD = Starting Delimiter (2 symbols)
FC = Frame Control (2 symbols)
DA = Destination Address (4 or 12 symbols)
SA = Source Address (4 or 12 symbols)
FCS = Frame Check Sequence (8 symbols)
DA = Ending Delimiter (2 symbols)
FS = Frame Status (3 symbols)

Figure A–4.
FRAME AND TOKEN FORMATS

as is the case for all fields covered by the FCS check, consists only of data symbols. Data symbols are not used in fields not covered by the FCS check.

The ED field of a frame is one delimiter symbol (T). It is followed by the frame status (FS) field that has a minimum of three control indicator symbols. These indicate whether the addressed station has recognized its address, if the frame has been copied or if any station has detected an error in the frame.

STATION ORGANIZATION

Figure A-5 shows the component entities necessary for an FDDI station. Identified components, conforming to both the IEEE 802 structure and the OSI concept of layering are: Station Management (SMT), which specifies the local portion of the system management application process, including the control required for proper operation of a station in an FDDI ring; MAC, which specifies the lower sublayer of the data link layer, including the access to the medium, addressing, data checking, and data framing; physical layer protocol (PHY), which specifies the upper sublayer of the physical layer, including the encode/decode, clocking and data framing for transmission; and physical layer, medium dependent (PMD), which specifies the lower sublayer of the physical layer, including power levels and characteristics of the optical transmitter and receiver, interface optical signal requirements, the connector receptacle footprint, the requirements of confirming optical fiber cable plants, and the permissible bit error rates. For the sake of simplicity, PMD is often considered part of PHY herein.

PHYSICAL LAYER (PHY) OPERATION

PHY provides the protocols and optical fiber hardware components that support a link from one FDDI station to another. PHY simultaneously receives and transmits. The transmitter accepts symbols from MAC, converts these to five-bit code groups and transmits the encoded serial data stream on the medium.

The receiver gets the encoded serial data stream from the medium, establishes symbol boundaries based on the recognition of a start delimiter and forwards decoded symbols to MAC. Additional symbols (QUIET, IDLE, and HALT) are

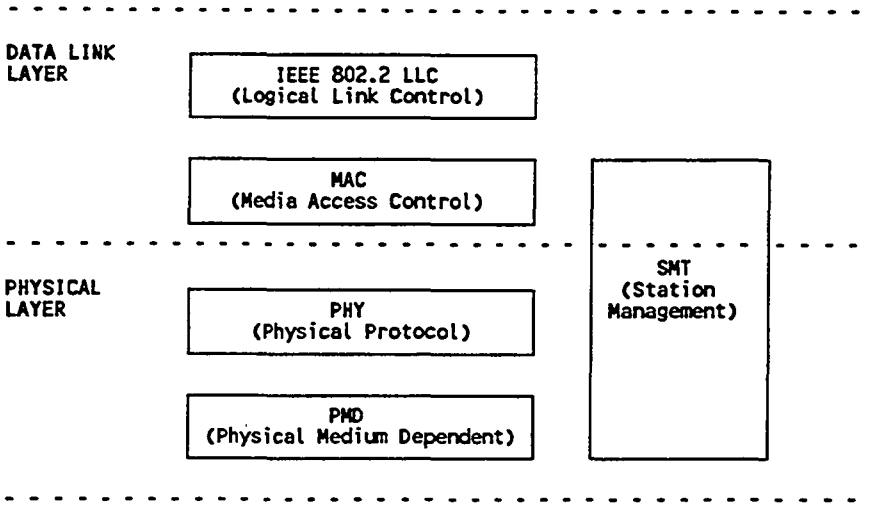


Figure A-5.
FDDI RELATIONSHIPS TO OSI MODEL

interpreted by PHY and used to support SMT functions.

SIGNAL CLOCKING

FDDI employs an interesting solution to the ring clocking problem. The total ring, including all of its stations and links, must remain the same apparent bit length during data transmission. Otherwise, some bits would lost or gained as a frame was repeated around the ring. In the face of jitter, voltage, temperature, and aging effects, such stability can only be realized through special provisions. PHY provides an elasticity buffer in each station. This elasticity buffer is always inserted between the receiver that employs a variable frequency clock to track the clock of the previous transmitting station, and the transmitter which runs on a fixed frequency clock. The elasticity buffer in each station is reinitialized during the preamble (PA) that precedes each frame or token. This has the effect of increasing or decreasing the length of PA that is initially transmitted as 16 or more symbols.

The transmitter clock has been chosen with 0.005 percent stability. With an elasticity buffer of 10 bits, frames of up to 4500 octets in length can be transmitted without exceeding the limits of the elasticity buffer.

TOKEN MAC FUNCTIONAL OPERATION

A major function of any station is deciding which station has control of the medium. Thus MAC schedules and performs data transfers on the ring.

The basic concept of a ring is that each station repeats the frame it receives to its downstream neighbor. If the destination address (DA) of the frame matches that MAC's address, the frame is copied into a local buffer and MAC notifies LLC (or SMT) of the frame's arrival. MAC marks any frame by setting indicator symbols in the FS field of a frame to indicate the recognition of its address, the copy of the frame or the detection of a frame in error. The frame propagates around the ring to the station that originally placed it on the ring. The transmitting station may examine the indicator symbols in the FS field to determine the success of the transmission. The MAC of this transmitting station is responsible for removing from the ring all frames that it has placed on the ring (a process called stripping). MAC recognizes these by the fact that the source address (SA) contained in them is its own address. IDLE symbols are placed on the medium during stripping.

If MAC has a frame from LLC (or SMT) to transmit, then it may only do so after a token has

been captured. A token is a special sequence that indicates that the medium is available for use. Priority requirements, necessary to assure the proper handling of frames, are implemented in the rules of token capture. Under these rules, if this station is not allowed to capture the token then it must repeat it (or in certain cases reissue a token) to the next. Having captured the token, by stripping it from the ring, MAC transmits a frame or frames, and when done issues a new token as notification that the medium is available for use by another station.

Frame Removal (Stripping)

Stripping is an important aspect of FDDI ring operation. At the instant when a station, having captured a token, initiates transmission, a busy ring may contain a number of frames in transit that were placed on the ring earlier by other stations.

These frames should all be removed from the ring by the respective stations that originally transmitted them before they arrive at the current transmitting station's receiver again. Thus, input to this station's receiver will consist of IDLE symbols and the remnants of stripped frames. In any case, while the station is transmitting, it will be removing from the ring anything that arrives at its receiver.

Remnants of stripped frames occur because a station's decision to strip a frame is based upon recognition of its own address in the SA field which will not occur until after the initial part of the frame has already repeated. These remnants cause no ill effects because various criteria, including recognition of an ending delimiter (ED), must be met before a frame is accepted as valid.

With the completion of transmission, the transmitting station continues to generate IDLE symbols following issuance of the new token until the SD of a new frame is received. At this time MAC reinitiates the examination of all arriving frames, stripping those that it has originated and repeating the others.

Timed Token Rotation (TTR) Protocol

FDDI uses TTR protocol to control access to the medium. Under this protocol each station measures the time that has elapsed since a token was last received. The initialization procedures establish the target token rotation time (TTRT) as that of the lowest bidding station. Two classes of services are defined. Synchronous service allows use of a token whenever MAC has synchronous frames queued for transmission. Asynchronous service allows use of a token only when the time since a token last was received has not exceeded the TTRT established. Multiple levels of priority for asynchronous frames may be provided within a station by specifying additional (more restrictive) time thresholds for token rotation.

The use of TTR protocol imparts some valuable ring operational characteristics. It allows stations to request and establish (via SMT procedures) guaranteed bandwidth and response time for synchronous frames. It establishes a guaranteed minimum response time for the ring because, in the worst case, the time between the arrival of two successive tokens will never exceed twice the value of TTRT. It also provides a guaranteed level of ring utilizations equal to (TTRT-RL)/TTRT, where RL is the physical ring latency – essentially the time for a token to propagate around the ring under no load conditions.

The choice of TTRT profoundly affects ring operation. Low values of TTRT (e.g. four milliseconds) may be used to establish an average token rotation time of four milliseconds and a guaranteed response time not exceeding eight milliseconds. This would be useful in a time-critical application (e.g. packetized voice). Larger values of TTRT may be used to establish very high ring utilizations under heavy loads. For instance, using TTRT of 50 milliseconds and a ring latency (RL) of 0.25 milliseconds, which is reasonable for a ring consisting of 75 stations and 30 km of fiber, a utilization of 99.5 percent is achieved.

STATION MANAGEMENT (SMT) OPERATION

SMT is the local portion of the system management application process, including the control required for proper operation of an FDDI station in an FDDI ring. The structure of an FDDI station and the relationship of SMT to other components in it are shown in Figure A-5.

SMT monitors activity and exercises overall control of station activity. These functions include control and management within a station for such purposes as initialization, activation, performance monitoring, maintenance, and error control. Additionally, SMT communicates with other SMT entities on the network for the purpose of controlling network operation. Examples of these SMT functions include the administration of addressing, allocation of network bandwidth, and network control and configuration.

The SMT connection management function (CMT) controls the logical interconnection of the PHY and MAC entities within a station, as well as establishing the physical connection between adjacent stations.

When SMT indicates a request to establish a connection with an adjacent station, CMT first establishes the existence of an operating physical link. To accomplish this, PHY - under control of CMT - sequences through a series of low level handshakes. These handshakes are accomplished through sequencing continuous streams of QUIET, HALT and IDLE symbols. A stream of QUIET symbols indicates that a link is physically disabled, due to either a link failure or deliberate shut down. A stream of HALT symbols, or an alternating sequence of HALT and QUIET symbols in the case of the master/slave concentrator connection, indicates link continuity, logically disables the connection and indicates the mode of the sender (master, slave or peer). A stream of IDLE symbols indicates willingness to establish a connection. Finally, a stream of IDLE symbols, followed by a JK indicates the acceptance of the requested physical connection.

Once a physical connection is established, CMT creates a logical configuration within the station by activating the appropriate paths between the PHY and MAC entities within that station. A large degree of flexibility is provided in the logical configurations which may be established, consistent with station functionality and desired station personality. This flexibility allows FDDI to support a wide range of topologies and applications. Two classes of stations and a concentrator are specified for use on an FDDI ring. A dual attachment station, shown in Figure A-6, has two PHY entities. It may attach directly to the ring, or indirectly via a concentrator. It may have one or two MAC entities. In the case of two MAC entities, one of these may be in each of the counter rotating rings, or both may be in the same ring. A dual attachment station may have optical bypass switches to remove it from both of the rings, at the same time healing them if the station is powered down or disabled by SMT.

Two classes of stations and a concentrator are specified for use on an FDDI ring. A dual attachment station, shown in Figure A-6, has two PHY entities. It may attach directly to the ring, or indirectly via a concentrator. It may have one or two MAC entities. In the case of two MAC entities, one of these may be in each of the counter rotating rings, or both may be in the same ring.

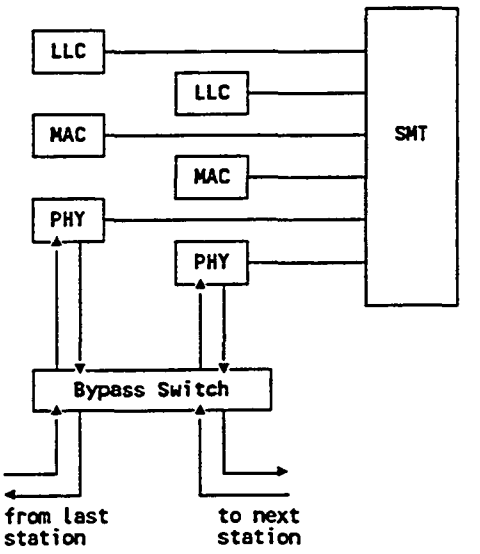


Figure A-6.
DUAL ATTACHMENT STATION

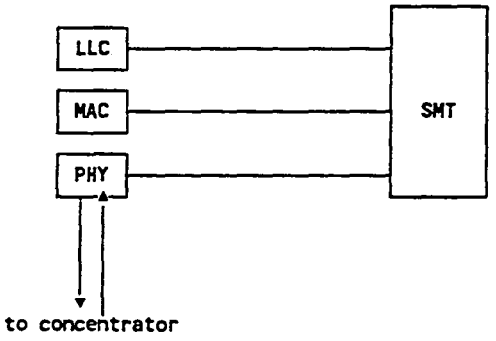


Figure A–7.
SINGLE ATTACHMENT STATION

A single attachment station (SAS) as shown in Figure A-7 has one PHY and one MAC. It may only be attached to the ring via a concentrator.

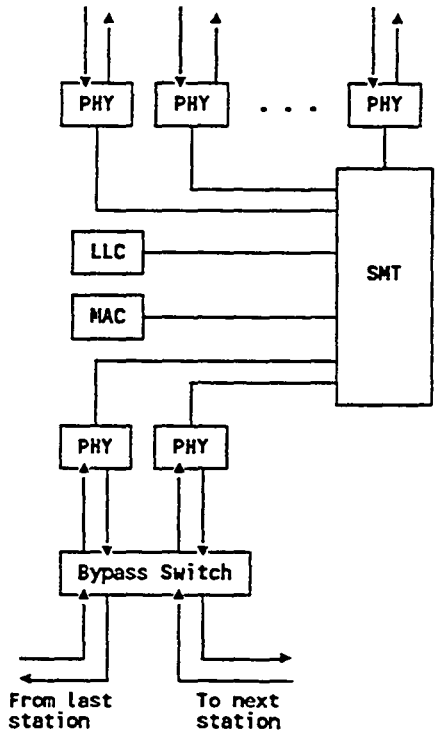


Figure A–8.
CONCENTRATOR

A concentrator, as shown in Figure A-8, has two PHY entities attaching it to the ring. It must have at least one MAC entity. A concentrator is used to connect single attachment (or dual attachment) stations into either of the two rings. It has the capability to bypass any failing single attachment station, thus healing the rings. It may also have optical bypass switches. Figure A-9 shows a portion of an FDDI ring composed of these elements. Stations 1 and 2 are dual attachment stations and station 3 is a concentrator providing attachment to the rings of stations 5 and 6, which are single attachment stations, and station 4. Station 4, even though it is a dual attachment station, must perform as a single attachment station insofar as its attachment to the ring via the concentrator is concerned. Operation of the second PHY of station 4 on some other ring is allowed, but there cannot be any interconnection of the two rings at a level visible to MAC or PHY.

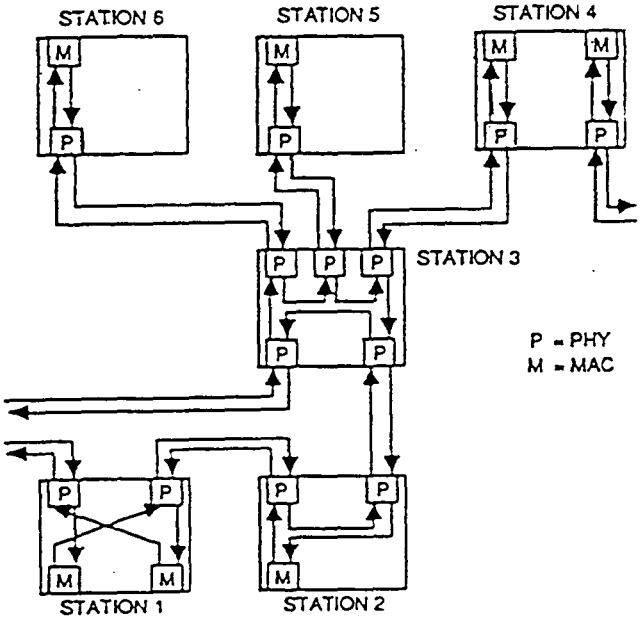


Figure A–9.
FDDI TOPOLOGY EXAMPLE

FDDI–II CONCEPTS

FDDI-II is an enhancement of the basic FDDI, or FDDI-I, as is commonly referred to. FDDI-I

offers a packet (switched) service. FDDI-II adds a circuit switched service.

A packet service is a service where the elements of data to be transferred are placed in packets. Packets are self defining in that each packet contains delimiters that mark its beginning and end. Packets may vary in length to accommodate the amount of data to be transferred. The delivery of a packet to the intended target station is a direct result of the target station recognizing its address which it finds at a specific location within the packet. More complex networks that require bridges to connect individual component networks, also use address-based mechanisms acting on the address (or addresses) contained within the packet. FDDI packets are called frames.

In contrast, a circuit switched service is a service that provides a continuous connection between two stations, or from one station to a number of stations. Instead of using addresses, the connection is established between the stations based upon some prior agreement. More complex networks that require bridges to connect individual component networks, likewise make the required connections on the basis of some prior agreement. These prior agreements may have been negotiated using packet messages or may employ any other suitable convention known to the stations involved in the transfer. This prior agreement may typically take the form of knowing the location of a time slot, or a number of time slots, that reoccur regularly relative to a readily recognizable timing market.

A common timing marker used in North America is the basic system reference frequency (BSRF) which is a 125-microsecond clock used by the public networks. Use of this clock will be assumed for FDDI-II. In local FDDI usage it is referred to as the cycle clock. Thus, in FDDI a circuit switched connection may be defined as N bits beginning at byte M (after the cycle clock marker) in wideband channel (WBC) number X. The last descriptor is necessary because FDDI offers 16 WBCs that may be independently assigned to either packet or circuit switched data. This definition allows connections at data rates of all multiples of eight Kbps (i.e. N = 1) up to the 6.144-Mbps data rate of a WBC. If need be, multiple WBCs may be used to accommodate higher data rates.

The data transferred in circuit switched mode is best described as a stream of data. The data rate is appropriate to the service being provided – with for example 64 Kbps being used for a digital voice-data stream. Other data stream rates, even up to many Mbps in the case of video, are used for other applications. Once a connection is established, the data rate remains constant.

The characteristics of packet and circuit switched data are of interest. Most packet data traffic occurs in random quantities and at random times. This is referred to as asynchronous traffic. Other packet traffic is more regular in nature – occurring in relatively predictable quantities on a regular time basis. This is referred to as synchronous (packet) traffic. This is contrasted with isochronous data. Isochronous data occurs in precise amounts on a precise time basis. It typically represents a sequence of digital samples from a sensor (e.g. voice or video). More importantly, isochronous data must be synchronized with clock information to ensure the accurate regeneration of the sampling clock (as distinct from the bit clock) to minimize distortion in data reconstruction. Isochronous data is more easily transferred in a circuit switched network.

Networks that carry isochronous data must do so synchronously, relative to the cycle clock. For the FDDI ring this means that one station (called the cycle master) must insert a delay for all isochronous data so that the ring appears to be an exact multiple of 125 microseconds in length. FDDI incorporates this delay in the cycle master in such a way that it does not cause any delay in the packet traffic. This is essential in providing an integrated services network with acceptable packet service.

FDDI–II OPERATION

Figure A-10, much like Figure A-5, shows how FDDI-II is to be implemented using one additional standard hybrid ring control (HRC). HRC becomes the new lowest sublayer of the data link layer, taking its place between MAC and PHY. Viewed in this manner, HRC provides the function of multiplexing data between the (packet) MAC and the isochronous MAC (not shown). This

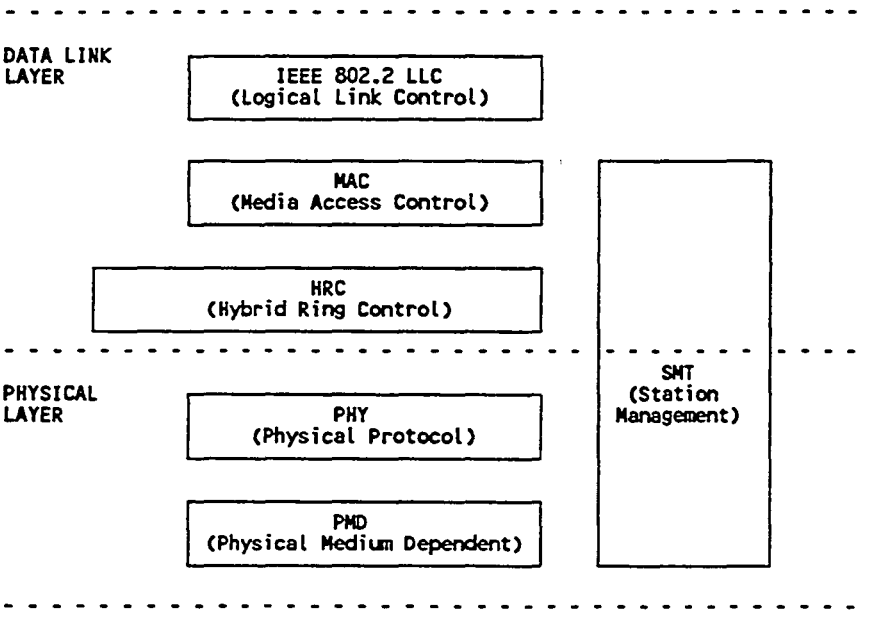


Figure A–10.
FDDI-II RELATIONSHIP TO OSI MODEL

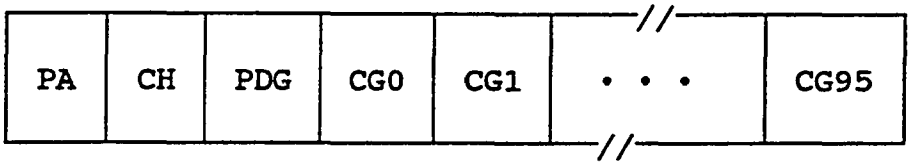
imposes the requirement on the (packet) MAC that it be able to transmit and accept data on a noncontinuous basis because packet data is interleaved with isochronous data.

FDDI-II is a network with 100 Mbps of bandwidth available. This bandwidth may be devoted totally to operation as a packet network or portions of this bandwidth may be separated for use of circuit switched channels. These portions of the FDDI-II bandwidth are dynamically assigned in units of WBCs. Up to 16 WBCs may be assigned to isochronous services. Each is 6.144 Mbps, which is four times the North American basic access rate and three times the European basic access rate. Each WBC is full duplex. These are independently allocatable and deallocatable. In effect, a broadband circuit capability has been provided with 16 available channels.

One of the strengths of FDDI-II is that once a station has been assigned a WBC, or a number of WBCs, that station may suballocate their combined bandwidth to itself or client stations on the basis of requirements. This suballocation may be in terms of any multiples of eight Kbps subchannels, including the commonly used 16-, 32-, 64- (B-channel), 384-, 1536-, 1920-, and 2048-Kbps subchannels. Mixtures of these data rates are allowed. If preferred, the aggregate of any or all of the allocated WBCs may be used as one virtual device, satisfying the needs of high-resolution video. Thus a multiplicity of virtual circuits may be provided.

Assignment of all 16 WBCs, each at 6.144 Mbps, multiplies out to a total of 98.304 Mbps. After allowance for synchronization overhead, if all of the WBCs were allocated, a residual one-Mbps channel for packet traffic is left. Data steering logic in the physical level hardware augments this with the bandwidth of any WBCs that are not assigned. This is accomplished in a linear manner, so that, for example, with eight (i.e. one-half of) the WBCs assigned to isochronous service, just over 50 Mbps of bandwidth would be available for packet traffic. The only impact to the packet traffic is the loss of the exact bandwidth of the allocated WBCs.

In operation, a ring is initialized in basic (token) mode and switches to a hybrid mode of operation, combining both packet and circuit switched data capabilities, only after a station has negotiated for and won the right to be cycle master along with the necessary synchronous




```
PA   = PREAMBLE (5 Symbols)
CH   = CYCLE HEADER (20 Symbols)
PDG  = Dedicated PACKET DATA GROUP (32 Symbols)
CGn  = CYCLIC GROUP 0 to 95 (32 Symbols per group,
       2 symbols for each of 16 possible isochronous
       channels)
```

Figure A-11.
CYCLE FORMAT

bandwidth allocation to support it. This cycle master then generates cycles at an eight-Khz rate (every 125 microseconds) and inserts the latency required to maintain an integral number of cycles synchronously on the ring. Alternatively, the ring may be initialized in hybrid mode.

A simplified cycle format is shown in Figure A-11. After the preamble (PA), a cycle header (CH), which begins with the JK starting delimiter, is used to carry control information and to assist in the allocation of each byte within the cyclic groups (CG0-CG95) to either the packet (token) data channel or one of the 16 possible WBCs. In effect, each of the 16 byte positions of the 96 cyclic groups ia available for assignment to the correspondingly numbered WBC (i.e. the 16 WBCs are byte interleaved within each cycle). If a WBC is not assigned, the corresponding data byte in each cyclic group is integrated as part of the packet (token) data channel.The 16-byte group designated as PDG is always dedicated to the packet (token) channel and guarantees a minimum bandwidth of 1.024 Mbps. A frame or token within this channel has the same frame format and interpretation as in the basic mode. JK cannot be used as the packet starting delimiter since it represents the cycle delimiter witch starts each FDDI-II cycle. Instead, a soft starting delimiter, unique in context, is used. The packet data channel includes the initial 16 bytes (PDG) and the bytes from each of the 96 data groups corresponding to the positions that are not assigned to WBCs.

The bandwidth required for both isochronous and synchronous traffic is allocated from the available FDDI bandwidth. Unallocated bandwidth is used on an as available basis for asynchronous traffic.

PRIORITY LEVELS

The basic FDDI, through the use of timed token rotation protocol, combined with the use of restricted and nonrestricted tokens, offers an extensive variety of packet data services. FDDI-II augments these with the capability to handle circuit switched data. Four kinds of traffic are defined. All may coexist in the same FDDI ring.

The highest priority is given to the circuit switched data. Isochronous traffic can be accommodated only by FDDI-II, and then only after a WBC has been assigned.

The second highest priority is given to synchronous packet traffic where predictable units of data are to be delivered at regular time intervals. Delivery is guaranteed with a delay not exceeding twice TTRT. This data may be transmitted following the capture of either a restricted or nonrestricted token.

The third highest priority is given to asynchronous traffic using restricted tokens that

may be transmitted upon the capture of either a restricted or nonrestricted token. Cooperating stations may only enter a restricted token mode of operation, which allows them to issue and use restricted tokens, after negotiating an agreement using nonrestricted tokens. Restricted token mode of operation allows stations to vie for the available synchronous bandwidth on a dialogue basis.

Lower priorities are given to asynchronous traffic that may be transmitted only upon the capture of a nonrestricted token. This mode of operation allows stations to vie for the available asynchronous bandwidth on single packet basis.

APPLICATIONS

Figure A-12 provides an overview of applications, showing equipment attachments both necessary and sufficient to determine FDDI-II requirements. This should be viewed as depicting the requirements of a typical customer site where high performance processors are installed. The figure assumes many instances of FDDI and each type of equipment and variations thereof. In all likelihood, no single instance of FDDI connects to all types of equipment shown, and certainly not to all the equipment of a large site. Also to be noted is that high traffic units, such as processors, may attach to multiple FDDI networks.In a packet switched mode, FDDI can be a backbone for bridges to a variety of other lower-speed LANs, for example, the various IEEE 802 media access methods. It may also provide a backbone for gateways to the public data networks. In both cases, connections to processors and mass storage subsystems can be provided. Connections to high performance workstations are likewise provided.

It is anticipated that FDDI-II implementations will follow those of the basic FDDI by about two years. It then follows that the rules of coexistence of these two FDDI implementations are important. Committee X3T9.5 has addressed this issue and is specifying FDDI-II as a perfect superset of the basic FDDI. This ensures interoperability, of FDDI-II implementations operating in the basic packet mode of operation with implementations of the basic FDDI. It also allows the use of FDDI-II chips for all applications once they become available.

FDDI STATUS

Standardization is a long and time-consuming process. After completion by ASC X3T9.5, any proposed standard is forwarded to ASC X3T9 for a technical letter ballot. This letter ballot approves forwarding it to X3, where a four-month public review is conducted, followed by an X3 letter ballot. It is then forwarded to the Board of Standards Review (BSR) for final review. Final editing is performed by an ANSI editor before publishing. This process is time consuming due to typical processing delays and the fact that any technical correction to the document requires its return to earlier stages.

MAC, first approved for X3 public review in February 1985, has just recently been approved as an American National Standard (ANS.139-1987). MAC is still in the ANSI editing process with publishing expected in the summer of 1987. The PHY document is just completing the X3 letter ballot with one comment to be resolved. Both MAC and PHY are draft international standard (DIS) with letter ballots due to be issued.

The PMD document was help up at the ASC X3T9.5 technical committee for resolution of connector issues. These have now been successfully resolved and PMD has been forwarded to X3 for the four-month public review. PMD is an International Draft Proposed Standard.

The SMT document, and increasingly the FDDI-II document, are now the main focus of activity of the ASC X3T9.5 committee. Committee X3T9.5 meets every two months with 85 to 105 attendees. The official voting membership, limited to one per corporation, exceeds 50. Working meetings are typically held in between the regular meetings. These working meetings are focusing on SMT and FDDI-II detail design definitions.

The FDDI standards effort has always been in step with the FDDI product implementations of a number of its supporters, including several semi-

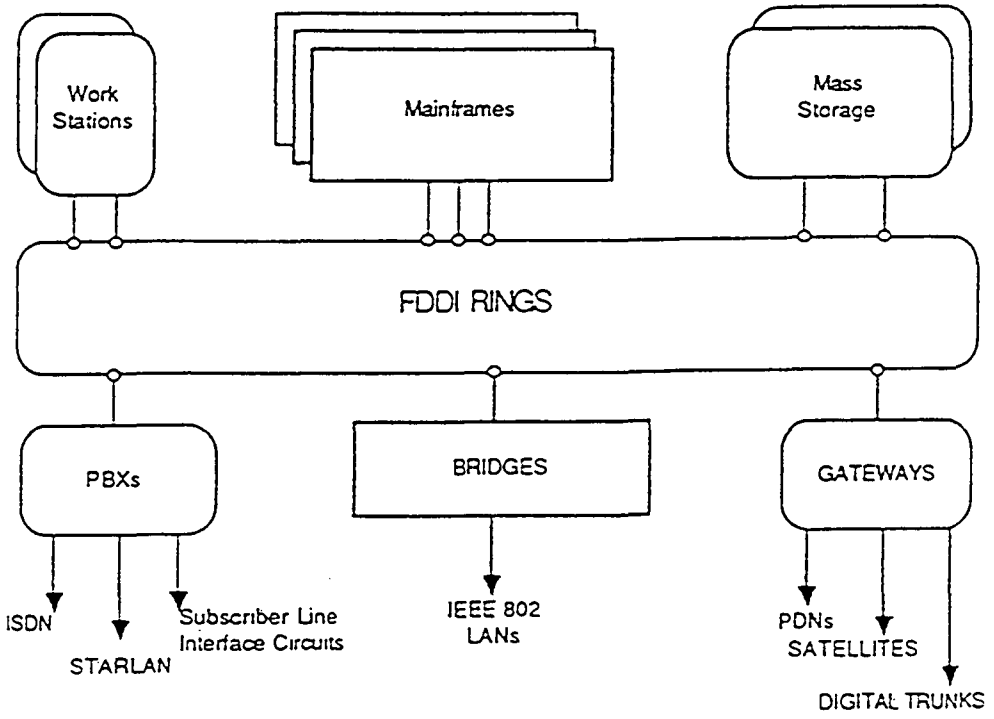


Figure 12
FDDI NETWORK EXAMPLE

conductor manufacturers working towards FDDI chip sets. It is appropriate that the press carried the announcement of the first FDDI product delivery early this year. Promised for delivery shortly is the first FDDI chip set which portents a more cost-effective FDDI.

CONCLUSION

The FDDI standards effort is proceeding well. Paralleling this standards development is the development of chip sets and components to make FDDI readily available as a product. With the increasing emphasis on standards, and the apt placement of FDDI as a high-speed LAN standard using optical fiber for a medium, FDDI is gaining recognition as the right solution for many diverse applications, including back end I/O networks, as a backbone network to medium-speed LANs, and as a front end network for high-performance work stations and graphic terminals. The coming of FDDI-II will extend FDDI functionality to embrace a number of circuit switched data applications, including voice and video, real time graphic displays, continuous sensor data streams, and distributed PABX network applications.

The author would like to thank James R. Hamstra for his contribution to this article.

REFERENCES

1. ANSI/IEEE Standard, 802.5-1085, Token Ring Access Method and Physical Layer Specifications.

2. American National Standard, FDDI Token Ring Media Access Control (MAC), ANS.139-1987.

3. Draft Proposed American National Standard, FDDI Token Ring Physical Layer Protocol (PHY), ASC X3T9.5 Rev. 14, Oct. 20, 1986.

4. Draft Proposed American National Standard, FDDI Token Ring Physical Layer Medium Dependent (PMD), ASC X3T9.5 Rev. 7, Feb. 20, 1987.

5. Floyd Ross, "FDDI – a tutorial," *IEEE Communications Magazine*, May, 1986.

6. Venkatramen Iyer and Sunil Joshi, "FDDI's 100-Mbps protocol improves on 802.5 spec's 4-Mbps limit," *EDN*, May 2, 1985.

7. Floyd E. Ross, "Rings are 'Round for Good," *IEEE Network Magazine*, January 1987.

8. M.J. Johnson, "Reliability Mechanisms of the FDDI High Bandwidth Token Ring Protocol," *Computer Networks & ISDN Systems*, Vol. 11, No. 2, pp. 121-131, 1986.

This page intentionally left blank.

Fiber Distributed Data Interface (FDDI) — A Tutorial

by Mark S. Wolter, National Semiconductor Corporation

Introduction

Some of the basic characteristics of the FDDI specification include fiber optic transmission media, ring topology, token access protocol and an architecture with a distributed management approach. The network parameters achieved by these characteristics include a 100Mbps data rate, 2-3 km distance between connections, up to 1000 connections on a network and a total distance of 200 km. The specification for FDDI is compliant with the Open Systems Interconnection.

Demand for FDDI

Mainframe computers were originally developed to support multiple users. As computers became more plentiful, the need arose to transfer data between computer systems, and proprietary computer networks were developed. The advent of mini, micro and personal computers led to the need to share expensive resources between several autonomous systems, each having their own operating environment. From this need, standardized *Local Area Networks* (LANs) evolved that supported shared printers, modems, and disk drives. With the availability of this standard network, new applications not suited to the available bandwidth of existing technology LANs were developed. The need for a high speed network, FDDI, was established. Shared data intensive resources such as file systems are now used as an integral part of a node's operating system and require fast access across a network. Plotters and graphic printers, and fast interactive support of high resolution graphic workstations require the solution offered by FDDI. Different networks, each with their own characteristics, were attached to each other, creating a nightmare for a network administrator responsible for network loading and fault isolation. Better management and greater bandwidth are needed for interconnection of existing departmental LANs onto a high speed FDDI backbone.

Fiber Optic Media

One of the characteristics of FDDI is the use of fiber optic cable as a communication media which has several advantages over alternative copper media. The bandwidth of coaxial cable or twisted pair wires limits the transmission speed and the distance between active connections as compared to fiber optic cables.

Fiber optic cable also has several advantages security over most alternatives. Fiber optic cable is immune to *Electromagnetic Interference* (EMI) so that it cannot be jammed by high power transmissions such as radio transmitters. Fiber optics do not emit *Electromagnetic Radiation* (EMR), avoiding the capability of eavesdropping. Splicing into a fiber optic cable for the purpose of eavesdropping is extremely difficult and

This article is reprinted from pp. 16 - 26 of *ConneXions — The Interoperability Report*, Vol 4., No. 10. October, 1990, with permission of Interop, Inc. *ConneXions — The Interoperability Report* tracks current and emerging standards and technologies within the computer and communications industry. It is published monthly by Interop, Inc., 480 San Antonio Road, Suite 100, Mountain View, CA, 94040, (415)941-3399.

can be easily detected as the receiving station as an increase in signal losses at the fiber optic receiver.

As technology improves, the cost of converting from one generation standard to the next can be expensive. Installation of a new transmission medium can be particularly expensive in the conversion costs. An investment into fiber optic cable can be used for FDDI now, as well as future networks using a gigabit per second technology.

Ring Topology

The topology of an FDDI network incorporates several features that allow for greater reliability and fault tolerance of a damaged network. The *dual counter-rotating ring* allows the network to be reconfigured around hardware failures that occur. Any single failure that occurs within a dual ring connection can be isolated from the ring and the integrity of the rest of the ring is still maintained. Multiple hardware failures on the dual ring will result in dividing the ring into separate rings that still function but are isolated from each other.

FDDI provides facilities which allow an FDDI network to be configured into a star topology. The support of a star topology in an office environment has several advantages to the ring topology. A typical office environment for a network consists of independent connections between each office workstation and a wiring closet, constituting a star topology. A wiring closet may be connected in a hierarchical manner to additional closets. This approach allows the flexibility to isolate stations from the network during installation and reinstallation of office equipment which occurs on a regular basis.

Other Topologies

An FDDI network allows for the support of several of these features available in a star topology. A station connected to the dual ring is knows as a *Dual Attach Station* (DAS) and supports the flexible reconfiguration mentioned previously. In a star topology, a station may be connected to a single ring and is known as a *Single Attach Station* (SAS). This SAS connection alone would not allow for the reliability available to a DAS, but with the constraint that it is used to attach to a *Dual Attach Concentrator* (DAC) the reliability of the dual ring may be maintained. This means that faulty SAS connections may be isolated by a concentrator, causing no damage to the remainder of the ring. *Single Attach Concentrators* (SAC) may be connected to DACs and other SACs to form a tree-structured hierarchical topology. This tree structure is connected to the trunk dual ring. (Figure A-13)

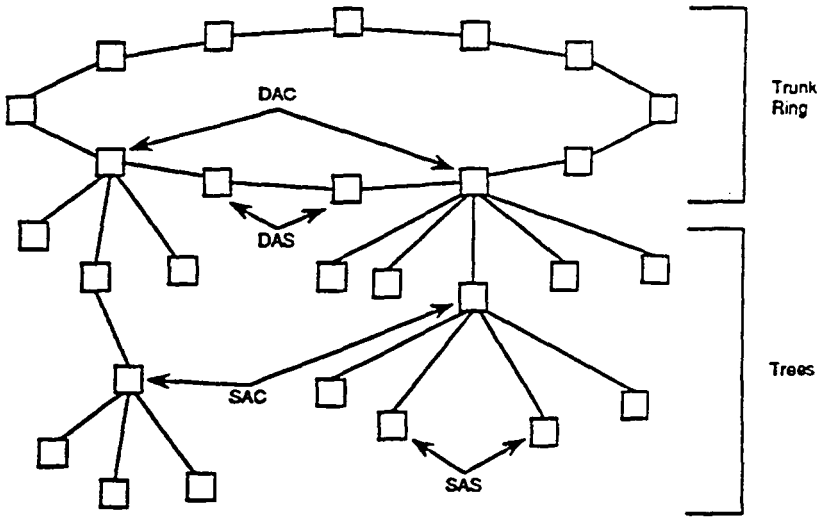


Figure A-13 . FDDI Topology

In the case of a catastrophic failure of a station on the ring trunk such as a power outage, the use of an optional *optical bypass switch* adds to the reliability of an FDDI network. Since this is a passive connection, it results in additional attenuation of the signal with the combined attenuation of the attached fiber segments. Care must be taken in the placement and use of this switch so as not to exceed the attenuation budget between FDDI stations.

Token Protocol

The two basic entities transmitted in an FDDI ring are called *frames* and *tokens*. Access to begin transmission on the ring is gained by capturing a special frame called a token. There is only *one* token allowed on the ring, and therefore only one station may transmit new data onto the ring at a time. A station transmits data by receiving a token, transmitting multiple frames until a timer expires, and retransmitting the token onto the ring. While frames circulate around the ring, they are copied by the receiving station. When the frame returns to the originating station, it is stripped by that station by not being re-transmitted.

The circulating token protocol provides an efficient access method to the ring, particularly at high network loads. It provides deterministic access to the network, guaranteeing eventual access to the network. The overall performance does not degrade as the number of stations added to the network increases.

Distributed Approach

A distributed approach was used in developing the FDDI network. The primary advantage of the distribution of network functions is that the loss of any single station does not inhibit the use of the network by the remaining stations. The design incorporates distributed clocking, fault recovery, and monitoring functions to add to the reliability and management capability of the network.

The use of independent clocks on each station adds to the reliability of the network. Since each station acts as a repeater, clock signals can be recovered from the upstream neighbor's signal transmission and re-transmitted by the station's local clock. Each station requires its own clock recovery circuit, but since there is local buffering and re-transmission there is no further need for the accommodation of accumulated jitter. If a clock were to fail at a given station, only that station would be affected and would be isolated by the neighboring stations on the ring.

The initialization process of an FDDI ring is done as a shared responsibility in which all stations take part. Prior to a ring becoming operational, all of the active stations participate in the parameter and token initialization. Timer expiration parameters must be set so that each station will be able to identify the failure of the ring due to the loss of a token. While the timer parameters are being set, the stations are also determining who will transmit the initial token frame. Since this approach allows any selected active station to recognize ring failure and to transmit the first token upon initialization or recovery, no single faulty station can cause the failure of the entire network.

A distributed approach is also used in the development of network management support. At the network interface level, all stations are peers and possess the same capabilities. Each station can transmit information about itself and its neighbors to a requesting station or as a broadcast message to all stations on the network. Each station can also request information and network reconfiguration. This flexibility allows for extensive diagnosis, isolation and reporting of problems on the network by any designated node at a given installation. The problems that can be identified include both physical failures and loading inefficiencies.

OSI Compliance

The FDDI standard is compliant with the lower layer of the OSI protocol stack. FDDI provides services specified by the Data Link and Physical layers. The FDDI specification includes the sublayers of the Physical layer called the *Physical* (PHY) and *Physical Media Dependent* (PMD) sublayers in FDDI terminology. The Data Link layer is subdivided into the *Link Layer Control* (LLC) and the *Media Access Control* (MAC) sublayers, of which FDDI specifies the MAC sublayer. The LLC sublayer is specified as an IEEE 802.2 standard, which interfaces directly to the MAC sublayer of FDDI as well as the other IEEE 802.x MAC sublayers of other network specifications.

PMD Layer

The PMD layer includes the specification for all of the transmission media hardware. The station-to-station attenuation budget of the fiber optic cables and connections are specified, as are the fiber optic transceivers. The optional optical bypass switch is also included. The fiber optic transmitter transforms an electrical signal into a signal suitable for driving a *Light Emitting Diode* (LED). The fiber optic receiver amplifies and filters the electrical output signal of a PIN (*p*-intrinsic-*n*) photodiode receiving light impulses.

MIC The cable connector receptacle defined by the FDDI standard is called a *Media Interface Connector* (MIC). The specification provides for the alignment of optical fibers. The fiber optic plug is not directly specified, but is an implied specification as a mate to the MIC. The receptacle can be mounted on a printed circuit board. The plug and receptacle are keyed to insure the interconnectability between conforming FDDI stations. MIC A (transmit on primary, receive on secondary) and MIC B (transmit on secondary, receive on primary) provide for attachment of DAS's and DAC's to the primary and secondary fibers of the dual ring. MIC M (*Master connection*) is used in a concentrator to provide an attachment for the MIC S (*Slave connection*) of a SAS. Optical losses are supplied by the fiber and receptacle vendors and must be considered in the installation plans.

The cable plant interface specifications include fiber types and attenuation. A 62.5μm core diameter fiber with 125μm cladding is the most commonly referenced fiber size, although 50μ/125μ and 100μm/140μ sizes are also referenced. The attenuation specification allows a -11.0dB loss between stations, including cable, splices, connectors, and fiber optic switches. Typically a 1300nm wavelength multimode cable is specified as less than 2.5dB/km attenuation. The maximum cable length is defined as the maximum distance possible without violation of the -11.0dB attenuation budget which includes connector loss. For 1300nm cable, this distance is around 2 km.

The PMD includes the specification for an optional optical bypass switch, which includes attenuation, interchannel isolation, switching time and media interruption time. When the station is powered off, the switch will go into bypass mode. Care must be taken to insure that the attenuation budget is not exceeded by too many optical bypass switches in bypass mode in series during a likely power down scenario.

PHY Layer The *Physical* (PHY) layer handles all of the symbol based functions. A *symbol* is the basic sequence of bits which represents data and control information. The PHY layer encodes and decodes the data and control symbols. It provides serial-to-parallel and parallel-to-serial conversions. It recovers the clock signal from the upstream neighbor, and includes an elasticity buffer for synchronizing to the local station clock.

NRZI Coding The PHY layer provides the 4B/5B symbol coding function and the NRZI bit coding function. The 4B/5B (four bit to five bit) coding maps data in four bit symbols to a corresponding five bit symbol that guarantees a logic 1 bit at least every five data bits in sequence to be transmitted. Since this five bit symbol is then transmitted using NRZI (non-return to zero, invert on ones) code, a signal transition occurs every time there is a logic one and is guaranteed at least every five data bits. This guaranteed transition is required by the clock recovery circuitry, a *Phase Locked Loop* (PLL), of the downstream neighbor to remain locked to the transmission frequency and recover the clock signal. The signal transitions are also required to maintain DC balance on the receive circuitry to minimize data noise.

The 4B/5B symbol coding and NRZI bit coding provide an efficient clock encoding scheme. The 4B/5B code uses one extra bit for every four bits of data to be transmitted, or 20% bandwidth overhead for clock encoding. This is why FDDI can transmit 100Mbps with a 125MHz clock frequency. A less efficient alternative encoding scheme might use Manchester encoding which guarantees at least one signal transition for every data bit to be transmitted, or 50% bandwidth overhead for clock encoding. The use of NRZI coding makes the highest frequency pattern or required bandwidth equal half the encoded frequency, or 62.5MHz for FDDI.

Since only about half of the possible 5B codes are mapped to corresponding 4B data patterns, several of the non-data codes are used for control symbols, under the constraint that they are also combinations that guarantee a logic 1 every five bits, with the exception of the Quiet symbol. These additional symbols are divided into four groups, being line state symbols, starting/ending delimiter symbols, control indicator symbols and violation symbols. Line state symbols include *Quiet* (Q), *Halt* (H) and *Idle* (I) symbols. Q indicates the absence of any transitions and loss of clock recovery ability. H indicates a forced logical break in activity while maintaining DC balance and clock recovery. I indicates normal condition between token and frame transmissions while providing the best case conditions for clock recovery. The starting delimiter consists of a unique *J* and *K* symbol pair and is used to designate the beginning of a frame. A pair of ending delimiters *Terminate* (T) symbols is used to terminate a token frame, while a single T followed by control indicator symbols terminates all other frames. Control indicator symbols consist of *Reset* (R) or *Set* (S) symbols and are defined by the MAC layer as well as implementation dependent extensions. Finally, all remaining symbol codes are designated as *Violation* (V) symbols, some of which may be recognized as an off-alignment H symbol.

The PHY layer also has to provide an elasticity buffer to allow for variation in clock frequency from one station to its upstream neighbor. If the upstream neighbor's clock frequency is higher, I's between frames will be dropped when repeating, and if the frequency is lower, I's will be added to the transmission stream.

Line States

The PHY layer generates control symbols and detects sequences of control and data symbols by entering into a designated line state. *Quiet Line State* (QLS) is entered after a stream of 16 or 17 Q's. *Master Line State* (MLS) is entered after a stream of 8 consecutive HQ symbol pairs. *Halt Line State* (HLS) is entered after a stream of 16 or 17 H's. *Idle Line State* (ILS) is entered after as many I's. *Active Line State* (ALS) is entered after a JK symbol pair frame starting delimiter. *Noise Line State* (NLS) is entered after 16 or 17 potential noise events have occurred without entering into another line state. Normal operating conditions of a network would toggle between ALS and ILS because frames and tokens will cause entry into ALS and the I's surrounding them will cause entry into ILS. Other line states are forced during the initialization process, the reconfiguration process or to signify a fault condition during the operation process.

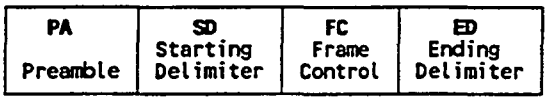
In order to prevent the propagation of bit stream errors and to simplify fault isolation, each PHY must provide a *repeat filter* that will not repeat V's or other NLS conditions to its transmitter. Although this would not be required in the PHY when a MAC is included in the repeat path because the MAC performs this function automatically, some configurations of a DAS or Concentrator provide paths that do not include a MAC.

MAC Layer The *Media Access Control* (MAC) layer handles all of the frame based functions and controls access to the transmission media. It handles the framing of data including the starting delimiter, frame class and ending delimiter. The MAC layer generates and recognizes source and destination addresses. It also generates and verifies a CRC frame check sequence.

Tokens The token is a unique "frame" that allows a given node to gain access to the ring for transmission of data. It consists of a preamble of 16 or more I's, a starting delimiter of a JK, a data field called the *frame control field* consisting of 2 symbols denoting that this frame is a token of a given class, and an ending delimiter of 2 T's. Two classes of tokens, *restricted* and *nonrestricted*, are defined. A nonrestricted token is used for normal operation, and a restricted token restricts all of the asynchronous (unreserved and unused) bandwidth of the network to be used by a specified set of two nodes for the remainder of their reserved connection. The MAC layer is responsible for capturing and re-transmitting the token. (Figure A-14).

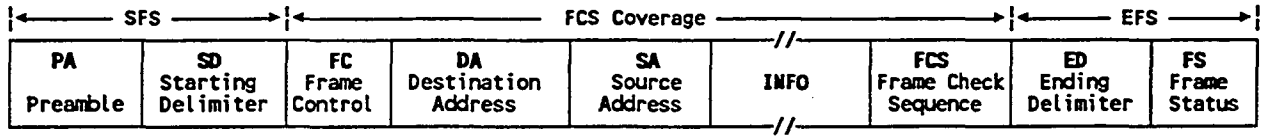
Frames Frames consist of the same preamble and starting delimiter sequence, a sequence of MAC control and data information, and an end of frame delimiter that consists of a single T symbol and a series of R and S symbols noting the status of the E, A, and C control indicator symbols as well as other implementation defined R/S indicators. The setting and resetting of the EAC indicators is administered by the MAC layer and designate whether a MAC layer condition has occurred indicating *frame check Error detected* (E), *destination Address recognized* (A) and *frame Copied* (C). The data sequence includes the frame control field, the destination and source address fields, the information field containing 0 or more symbols, and 8 symbols of the frame check sequence field. (Figure A-14)

TOKEN FORMAT:



PA = Preamble (16 or more I symbols)
SD = Starting Delimiter (1 JK symbol pair)
FC = Frame Control (2 symbols)
ED = Ending Delimiter (2 T symbols)

FRAME FORMAT:



SFS = Start of Frame Sequence
PA = Preamble (16 or more I symbols)
SD = Starting Delimiter (1 JK symbol pair)
FC = Frame Control (2 symbols)
DA = Destination Address (4 or 12 symbols)
SA = Source Address (4 or 12 symbols)
INFO = Information (0 or more symbols)
FCS = Frame Check Sequence (8 symbols)
EFS = End of Frame Sequence
ED = Ending Delimiter (1 T symbol)
FS = Frame Status (3 or more R or S symbols)

Figure A-14 . Token and Frame Formats

The frame control field denotes the type of frame. A void frame is a frame whose content is ignored while it is stripped by the transmitting station, its use is explained later. MAC frames, SMT frames, and LLC frames are used to transmit information between matching layer entities between stations. A reserved frame exists for use for the dependent implementation of a network.

MAC frames are used to initialize several MAC parameters including those explained below. Special MAC frames include the *Beacon Frame* used to localize the fault of a serious ring failure and the *Claim Frame* used to determine which station will transmit the first token and initialize the ring.

Station Management (SMT) functions implement network management facilities and use their own class of frames. A special SMT frame exists that addresses the next station, and is used by SMT for ring mapping and recovery functions. LLC frames are used for the transmission of data between the upper layers of the OSI stack and serve as the main data stream for network connectivity. LLC frames can be sent as synchronous or asynchronous frames. Synchronous frames are frames that are sent as part of a station's predetermined guaranteed bandwidth and are top priority frames. Asynchronous frames use the unreserved or unused bandwidth of the network and allocation of this bandwidth is provided to the stations in a round robin fashion.

Addressing

The destination and source address fields can be either 16 or 48 bit addresses. All stations are required to fully support the 16-bit address frames, repeat all 48-bit address frames and support the 48-bit Claim, Beacon and Broadcast address frames. Broadcast addresses consist of

all ones. Null addresses of all zeros are not recognized by any station as its address. The first bit of the destination address field denotes whether a frame is an individual or group address, while the second bit of the 48 bit address denotes whether the address is a universally or locally administered address. The first bit of the source address always denotes an individual address.

The information field consists of 0 or more eight bit octets of data transmitted as symbol pairs. The frame check sequence field consists of a 32 bit CRC polynomial commonly used for communication protocols that encapsulates the entire data sequence, including the frame control, addressing and information fields, and the frame check sequence field itself.

Timers

Three timers and a counter are used by the MAC layer to regulate the operation of the ring. The *Token Holding Timer* (THT) controls how long the station may transmit asynchronous frames. The *Valid-Transmission Timer* (TVX) is used to recover from transient ring error situations. It times the delay since the last valid transmission, and a void frame may be used to reset the TVX in the absence of a normal data frame. The *Token Rotation Timer* (TRT) controls ring scheduling during normal operation and to detect and recover from serious ring error situations. The *Late Counter (Late_Ct)* counts the number of TRT expirations since the MAC was reset or a token was received.

Three additional counters are used to aid in problem determination and fault location. The *Frame_Ct* is a count of all complete (non-fragment) frames received. The *Error_Ct* is a count of frames received with a frame check error and without the E indicator already set. The *Lost_Ct* is a count of all instances that while receiving a frame or token an error occurred that threatened the integrity of the frame itself, such as the reception of a V symbol.

Claim Process

A station with data to transmit gains access to the ring by capturing a token. It continues to transmit multiple frames until it is finished or its available time expires, and then re-transmits the token onto the ring. While frames circulate around the ring, they are recognized by the receiving station which sets the A indicator.

If the station can, frames are copied to that station's available buffer space after which the station also sets the C indicator of the frame during re-transmission. When the frame returns to the originating station, it is stripped by that station by transmitting I's immediately following its recognition, i.e., the source address field.

During the Claim process, all stations in a ring agree upon a common *Target Token Rotation Time* (TTRT), the station with the lowest bid TTRT winning. Some stations are also assigned a synchronous allocation time that guarantees that station an allotted transmission bandwidth. A station's synchronous allocation time is based on that ring's implementation and set by an SMT-to-SMT communication. Each time a token is received, *Late_Ct* is reset to zero and TRT is reset to TTRT and begins to count down. If another token is received before TRT expires, both

asynchronous and synchronous transmission may take place. At this time, THT is set to TRT and TRT is reset to TTRT to begin timing the next rotation time. THT is then the unreserved and unused bandwidth that is available for asynchronous transmission, and asynchronous frames may be transmitted until THT expires. An asynchronous priority scheme also exists that requires THT to be of greater value than a threshold value *T_Pri(n)* before a frame of priority level (n) may be transmitted.

If instead TRT expires before the arrival of another token, *Late_Ct* is incremented to one and only Synchronous transmission can take place for its allocated time upon reception of the token. If TRT expires a second time before receiving a token, *Late_Ct* is incremented to two, the token is considered lost, and a new Claim process will begin. This protocol guarantees an average TRT (or average synchronous response time) not greater than TTRT, and a maximum TRT (or maximum synchronous response time) not greater than twice TTRT. This protocol also guarantees that the asynchronous bandwidth is not entirely taken by one station, but that every other station has a chance to use this bandwidth in a round-robin fashion before a given station gets another chance.

SMT Specification

The FDDI standard also includes a *Station Management* (SMT) specification which allows for the necessary network management functions for performance monitoring, fault detection, and error recovery. SMT is subdivided into three areas: *Frame-Based Management*, *Connection Management* (CMT) and *Ring Management* (RMT). SMT provides frame-based functions that gather information about and exercise control over the FDDI network. CMT manages the PHY components and their interconnections, and uses MAC and PHY entities within a station to achieve logical attachment of the station to the ring. MAC layer components and the rings to which they are logically attached are managed by RMT. (Figure A-15)

Management Frames

SMT frame-based management services include the use of several types of SMT frames for performing various functions. *Neighborhood Notification* (NN) uses *Neighborhood Information Frames* (NIFs) to determine its *Upstream and Downstream Neighbor Addresses* (UNA and DNA), to provide supplemental duplicate address detection, and to verify the operation of local MAC receive and transmit paths in the absence of any other traffic. NIFs provide information for resolving network faults and constructing logical ring maps. *Status Report Frames* (SRFs) provide optional information by reporting network parameter conditions and connection events.

Optional *Parameter Management Frames* (PMFs) are used by a protocol to operate on all SMT *Management Information Base* (MIB) attributes to allow remote management of station attributes, such as the synchronous bandwidth allocation of each station. *Station Information Frames* (SIFs) provide station status obtained remotely by polling stations for connection and configuration parameters, and statistical operation information. *Echo Frames* provide for SMT-to-SMT loopback testing on a ring. *Root Concentrator Polling* allows a node in a concentrator tree to determine where it is within a tree of concentrators and where it resides in terms of

global connectivity by using any of the SMT request/response protocols. *Extended Service Frames* (ESFs) are defined for extending new SMT frame based protocols and require a unique identification parameter. *Request Denied Frames* (RDFs) are defined to allow backward compatibility by allowing new protocols to easily identify incompatible stations in a mixed environment.

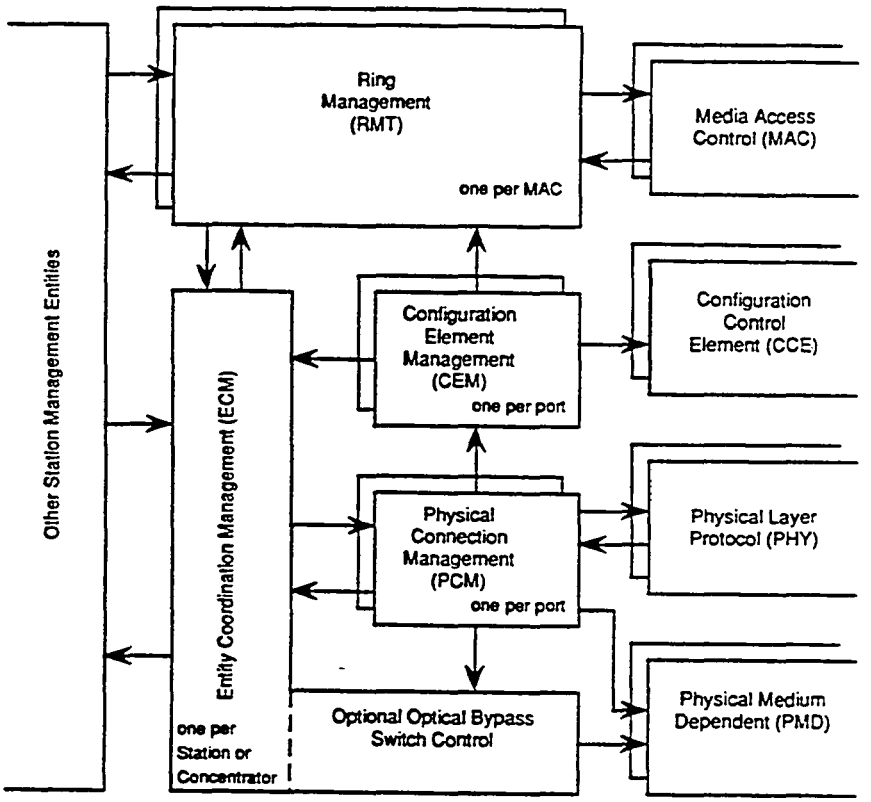


Figure A-15 . Station Management Interfaces

Connection Management

Connection Management (CMT) operates the insertion and removal of PHY and PMD entities called *Ports* to the ring, and the connection of ports to the MAC entities. CMT is further subdivided into three areas, including ECM, PCM, and CFM. *Entity Coordination Management* (ECM) is responsible for the media interface to the FDDI network, including the coordination of the activity of all the Ports and the optional optical bypass switch associated with that station or concentrator. After completing optical bypass switching, ECM signals *Physical Connection Management* (PCM) when the media is ready to begin initialization of the PHY. ECM also coordinates the *Trace and Path Test* functions, which are used to localize a *Stuck Beacon* condition and test the path to the nearest upstream MAC, respectively.

PCM initializes the physical connection between the Port being managed and another Port, such as in an adjacent station or concentrator on the FDDI ring, by signaling between these Ports upon a request from the

ECM. Signaling is done by transmitting a continuous stream of symbols until a neighbor responds, which transitions to a new line state and the station begins to transmit a different stream of symbols awaiting the next response. PCM sequences through a number of bit signals to communicate such connection information as the port type (A,B,M,S), the compatibility of this connection, the duration of a *Link Confidence Test* (LCT), the availability of a MAC for LET, the failure of a LET, the assignment of a MAC for Local Loop testing, and the assignment of a MAC to this port for ring operations as a tree or peer MAC. Once the connection has been verified completely by this procedure, a response is issued to *Configuration Management* (CFM). Through the use of this signaling feature, CMT on one station can force its neighboring CMT to a known state by transmitting a series of symbols, aiding in the isolation of faults in a ring.

Configuration Management

Each Port in a station or concentrator has a PCM entity associated with it. Each PCM has a *Configuration Element Management* (CEM) entity associated with it. Each CEM controls an associated *Configuration Control Element* (CCE), sometimes referred to as the configuration switch, which physically connects the MAC and Port entities. *Configuration Management* (CFM) collectively refers to all of the CEMs in a station or concentrator and manages the configuration of the MAC and Port entities therein. CFM supports the configuration of all types of stations and concentrators including DAS, SAS, DAC, and SAC through the use of four different CEMs supporting A and B connections to a dual attach ring and the S and M connections for all single attachments. There is also a single CEM specifically used for coordinating the connection of all MACs within a node called *MAC Placement Management*. Each CCE is required to provide a primary path connection to its associated Port, and may optionally include the capability for connecting the secondary path. The allowance of a local path that may be used for connecting a dedicated MAC to the Port also exists. CFM is also responsible for ensuring that a reconfiguration of the ring does not create any undesirable frames on the ring. This includes *ring scrubbing*, or the removal of frames from MACs that are no longer of the token path, non-concatenation of frames, or sourcing at least 16 I's before connecting a new path into the ring, and the avoidance of concurrently processing A and B Port CEMs which may result in the entrance into undesirable states.

Ring Management

Ring Management (RMT) is responsible for receiving status from the MAC and CMT and reporting the status of the MAC to SMT. RMT identifies a Stuck Beacon by the expiration of a timer, and then *Directed Beacons* are sent to see if the problem still remains unresolved. If it does, RMT initiates the Trace function which uses PHY signalling to identify the fault domain, which results in a Path Test in all nodes in the fault domain. A number of conditions can be used by RMT to detect duplicate addresses during the Claim and Beacon processes which could result in the processes never being resolved. Resolution of duplicate address problems that prevent *Ring_Op* is done so that MACs with duplicate addresses will not prevent communication by the entire FDDI ring. This can be done by changing the MAC address, configuring the MAC to lose the Claim process and disabling the LLC services, or removing the MAC with the duplicate address from the ring.

Summary of Benefits

FDDI is a fast reliable network protocol which provides many unique network management functions. Its fiber optic transmission media allows longer, more secure connections between stations. Its efficient use of the available bandwidth insures maximum throughput. Its token method provides deterministic access services which will not degrade as the number of connections increase. It provides advanced capabilities with provisions made for extension to the protocol.

References

[1] American National Standards Institute (ANSI), "FDDI Token Ring Media Access Control (MAC)," American National Standard, ASC X3T9.5, 1986

[2] ANSI, "FDDI Physical Layer Protocol (PHY)," Draft Proposed American National Standard, ASC X3T9.5, Rev. 13, 1986

[3] ANSI, "FDDI Physical Layer Medium Dependent (PMD)," Draft Proposed American National Standard, ASC X3T9.5, Rev. 7.3, 1988

[4] ANSI, "FDDI Station Management (SMT)," Working Draft Proposed American National Standard, ASC X3T9.5, Rev. 6.1, 1990

This page intentionally left blank.

APPENDIX B
DATA STRUCTURES

This appendix contains data structures which are useful in understanding the RC (Report/Command) Interface, including both generic and 4211-specific aspects of RC. Aspects of the interface which are specific to the 4211 are described in this manual. Generic RC data structures and commands are documented in the *Common Boot and RC Interface User's Guide*.

```
/*
 * @(#) if_prg.h 4.10(#)
/*
                if_prg.h

        Definitions for the Interphase 4211 Peregrine device driver.
*/
#ifndef _IF_PRG_H
#define _IF_PRG_H
#define FDDIMTU 4096+256        /* Per RFC1103 */

/*
    Number of microseconds to wait after issuing a board reset.
*/
#define PRG_RESET_WAIT   400000
/*
    RFC1103 defines a hardware type code for ARP messages.  This
    has not filtered out to all the ARP code in the world.  We define
    it here.
*/
#define ARPHRD_FDDI     6
/*
    When updating a segment number the segment offset must first be
    set to indicate an update is in progress.  This prevents the board
    from reading an invalid pointer.
*/
#define RC_UPDATE_IN_PROGRESS                           0xffff

#define BD_SHORT_IO_SIZE                                  512
#define RC_CHANNEL_CONTROL_OFFSET                          16
#define RC_RESERVED_SPACE           (RC_CHANNEL_CONTROL_OFFSET)
#define RC_MAX_NUMBER_OF_CHANNELS    \
        ((BD_SHORT_IO_SIZE - RC_RESERVED_SPACE) / sizeof(EX_RC))


/*
 * These are the data types for the V/FDDI 4211 Peregrine.
 */

#ifndef _TYPES_H_
#define _TYPES_H_

typedef     char                    byte_t;
typedef     unsigned char           ubyte_t;
typedef     short                   short_t;
typedef     unsigned short          ushort_t;
typedef     short                   word_t;
typedef     unsigned short          uword_t;
typedef     int                     int_t;
typedef     unsigned int            uint_t;
```

```
typedef     long                    long_t;
typedef     unsigned long           ulong_t;

typedef     char                    i8;
typedef     unsigned char           u8;
typedef     short                   i16;
typedef     unsigned short          u16;
typedef     int                     i32;
typedef     unsigned int            u32;
typedef     long                    l32;
typedef     unsigned long           ul32;

typedef     char                    BYTE;
typedef     unsigned char           UBYTE;
typedef     short                   SHORT;
typedef     unsigned short          USHORT;
typedef     short                   WORD;
typedef     unsigned short          UWORD;
typedef     int                     INT;
typedef     unsigned int            UINT;
typedef     long                    LONG;
typedef     unsigned long           ULONG;

typedef     volatile char           hw_i8;
typedef     volatile unsigned char  hw_u8;
typedef     volatile short          hw_i16;
typedef     volatile unsigned short hw_u16;
typedef     volatile int            hw_i32;
typedef     volatile unsigned int   hw_u32;
typedef     volatile long           hw_l32;
typedef     volatile unsigned long  hw_ul32;

#define TRUE    1
#define FALSE   0
#define true    1
#define false   0

#define LOWORD(x)    (USHORT)((UINT)(x) & 0xffff)
#define HIWORD(x)    (USHORT)((UINT)(x) >> 16)
#define W_SIZ(x)     (sizeof(x)/sizeof(SHORT))

typedef int             (FUNC)();
typedef int             (*pfi_t)();
typedef u32             (*pfu_t)();
typedef void            (*pfv_t)();
typedef unsigned int    boolean;

#define BIT(x)  (1 << x)
#define reg     register

#define A_CNT(arr)      (sizeof(arr)/sizeof(arr[0]))
#define A_END(arr)      (&arr[A_CNT(arr)])
#define A_LAST(arr)     (&arr[A_CNT(arr)-1])

#define nil(x)      ((x)0)
#define nilp(x)     ((x *)0)
#define max(a,b)    ( (a > b) ? a : b )
#define min(a,b)    ( (a < b) ? a : b )

#endif  /* _TYPES_H_ */
```

```
/*
    External view of RC channel control structure
*/
typedef volatile struct ex_rc_s {
    u16 reserved1;      /* Reserved word            */
    u16 intr_timer;     /* Interrupt/timer word     */
    u16 r_segment;      /* Read Segment number      */
    u16 r_offset;       /* Read offset value        */
    u16 w_segment;      /* Write Segment number     */
    u16 w_offset;       /* Write offset value       */
    u16 reserved2;      /* Reserved word            */
    u16 reserved3;      /* Reserved word            */
} EX_RC, * EX_RCP;

/*
    The format of short io space as seen by the host.
*/
typedef volatile struct erc_s {
        u16     cint;           /* Hardware control field (controller-specific)      */
        u16     icid;           /* Controller writes 0x20 here at RC initialization  */
        u32     reserved1;
        u32     reserved2;
        u32     reserved3;
        EX_RC   rc[RC_MAX_NUMBER_OF_CHANNELS];
} E_SHIO, *E_SHIOP;
/*
    Reading and writing channel pointers must be done carefully due to
    the fact that the board may be updating them.  The following macros
    should be used to read and write these pointers.
*/
/*
    This macro updates the write pointer for a channel.  It is
    passed the address of the segment number field, the new segment
    number, the address of the offset field and the new offset value.
    This macro should be used when updating a b2h channel read pointer.
*/
#define UPDATE_CHANNEL_PTR(seg,newseg,offset,newoffset) { \
    if (*(u16 *)seg != (u16)newseg) { \
        *(u16 *)offset = (u16)RC_UPDATE_IN_PROGRESS;\
        *(u16 *)seg = (u16)newseg;\
        *(u16 *)offset = (u16)newoffset;\
    } \
    else \
        *(u16 *)offset = (u16)newoffset;\
}


/*
    This macro reads an RC channel pointer.  We must be careful when
    reading such a pointer because the pointer may be in the process
    of being updated by the board.  It is passed the address of the
    the segment number to read, the address of the offset to read and
    it returns the segment number and the offset value.  This macro should
    be used when reading a b2h write pointer or h2b read pointer.
*/
#define READ_CHANNEL_PTR(saddr,oaddr,seg,offset){ \
    u16 stmp; \
    do{ \
        do{ \
            (u16)seg = *(u16 *)saddr;\
            (u16)offset = *(u16 *)oaddr;\
        } while (offset == RC_UPDATE_IN_PROGRESS); \
        stmp = *(u16 *)saddr; /* insures that new seg number is read in */ \
    } while (stmp != seg); \
}
```

```
/*
    Update host-resident image of the segment offset.  The segment
    number either did not change or was updated in the do_h2b_link_dir()
    function.

    Update shortI/O version of the segment offset.  This actually
    notifies the board that the command is there, otherwise it
    will not be processed.
*/
#define update_h2b_ptr(len, pi, cn) { \
        pi->prc[cn].w_offset += len; pi->shio->rc[cn].w_offset += len; \
}
/*
        For each segment of a channel a segment structure is kept.
*/
typedef struct  prg_seg_desc {
        caddr_t addr;           /* Virtual address of segment.   */
        caddr_t paddr;          /* Physical address of segment.  */
        ushort  segno;          /* Segment number                */
} HSD, *HSDP;
/*
        Size of a channel segment
*/
#define PRG_CHANNEL_SEG_SIZE       2048
/*
        We use 3 channels, 2 host to board, 1 board to host.
*/
#define CHANNELS_PER_BD            3
#define SEGS_PER_CHANNEL           6
/*
        Total number of segments.
*/
#define SEGMENTS_PER_BD        (CHANNELS_PER_BD * SEGS_PER_CHANNEL)
/*
    Number of transmit channels per board.
*/
#define XMIT_CHANNELS_PER_BD 1
/*
    Number of h2b segments per board.
*/
#define H2B_SEGS_PER_BD      (XMIT_CHANNELS_PER_BD * SEGS_PER_CHANNEL)
/*
    The count of the number of response channels.
*/
#define RESP_CHANNELS_PER_BD 2
/*
    Total number of b2h segments per board.
*/
#define B2H_SEGS_PER_BD  (RESP_CHANNELS_PER_BD * SEGS_PER_CHANNEL)
#define XMIT_CH     0   /* index of inhost transmit channel descriptor        */
#define RCV_CH      1   /* index of inhost receive channel descriptor         */
#define RESP_CH     2   /* index of inhost response channel descriptor        */
typedef struct hostChannelDesc
{
        EX_RCP          channel;    /* ptr to channel descriptor in shortio   */
        u16             chid;       /* value of the channel id                */
        u16             intr_timer; /* min milliseconds till interrupt        */
        u16             r_segment;
        u16             r_offset;   /* "internal" version of the read ptr     */
        u16             w_segment;
        u16             w_offset;   /* "internal" version of the write ptr    */
        caddr_t         va;         /* Read ptr virtual addr for response     */
                                    /* channels                               */
} HCD, *HCDP;
```

```
/*
    This macro converts from the channel id passed up from the
    4211 to a pointer to a channel descriptor in short io.  It is passed
    a pointer to the info struct, the offset passed up from the board,
    and returns an EX_RCP.

    EX_RCP                  Output: Pointer to channel descriptor
    B2H_CHID(p,bc)
    struct prg_info *p;     Input: Pointer to info struct
    u16             bc;     Input: Channel id from 4211
*/
#define B2H_CHID(p,bc)  ( (EX_RCP)((u32)p->shio + (u32)bc) )
/*
    This macro converts from channel descriptor to a descriptor for
    use by the 4211.  It is passed a pointer to the info struct, the pointer
    to the channel descriptor, and returns an offset within short io.

    u16                     Output: Channel id for 4211
    H2B_CHID(p,hc)
    struct prg_info  *p;    Input: Pointer to info struct
    EX_RCP           hc;    Input: Pointer to host channel descriptor

*/
#define H2B_CHID(p,hc)   ( (u16)((u32)hc - (u32)p->shio) )

/*
    This macro converts a pointer to a channel descriptor (EX_RCP) in short io
    to host segment number.  This can then be used to index into the segs array
    within the prc array in the info struct.  It is passed a pointer
    to the info struct and the channel descriptor pointer and returns an
    integer channel number.
    u32                     Output: Host channel number.
    RCP2HCHID(p,r)
    struct prg_ifo   *p;    Input: Pointer to info struct
    EX_RCP           r;     Input: Address of channel descriptor in short io
*/
#define RCP2HCHID(p,r)   ( (((u32)r - (u32)&p->shio->rc[0])\
                                / sizeof(EX_RC)) )
/*
        This value is placed in the intr_timer field to disable the timer
*/
#define RC_DISABLE_TIMER        0xffff
/*
    Frequency at which we get timer interrupts from the board on response
    channels.  This is the minimum interrupt interval, only get interrupted
    if there is work queued on the channel.  Value is in units of 100
    microseconds.
*/
#define PRG_RESP_INTR_TIMER      0
/*
    Default amount of time to wait for the board to respond to a command
    when not using interrupts.  Value is in microseconds.
*/
#define RC_DEF_WAIT  2000

/*
    Receive buffer list scheme.

    You declare three rcvBufferDesc items, one for each of small, medium,
    and large buffer lists.
    You malloc a page of memory from the kernel for each RBD which will
    hold the buffer lists, and then record the physical address and
    virtual address of each list in the "rbp_phy_addr" and "rbp"
    elements of their RBP's.
    You assign a value for the size of the buffers each list will
    point to, and put that in the "rbufSize" field.
    You calculate the number of entries which can go in the lists,
```

```
    and put that in the numEntries field of each.  Be sure to leave
    room for the delimiter element.
    You calculate the address at which the "wptr" should wrap to the
    beginning (let's you do this arithmetic only once).
    Finally you work though each list and allocate receive buffers of
    the appropriate size for each element, recording both the mbuf
    pointer (tag) and the physical address (bufAddr).
*/

typedef struct rcvListEntry
{
        u32     bufAddr;   /* physical address of the receive buffer          */
        u32     tag;       /* pointer-to-mbuf or virtual address of buffer    */
} RLE, *RLEP;

typedef struct rcvBufferDesc
{
        RLEP    rbp;            /* virtual address of the buffer list         */
        u32     rbp_phy_addr;   /* keep the phy addr of buffer list also      */
        RLEP    wptr;           /* write pointer to buffer list               */
        RLEP    wrapAddr;       /* address at which wptr should wrap around   */
        u16     numEntries;     /* number of list elements                    */
        u16     rbufSize;       /* size of each buffer in bytes               */
} RBD, *RBDP;

#define RBD_END_SENTINEL        0xffffffff
/*
        bufhdr flag definitions.
*/
#define PB_FREE         0001            /* Buffer is free */
/*
        Size of receive and transmit buffers.
*/
#define PRGBUFSIZE      5000


/*
        Number of receive buffers to allocate.
*/
#define REC_BUFS_PER_BD    48
#ifdef sun
/*
        Number of transmit buffers to allocate.  Used only on a SUN.
*/
#define XMIT_BUFS_PER_BD   16
#endif /* sun */

/*
        Debug macros and defines.
*/
/*
    Layout of a 32 bit word in short IO space after the controller has
    been reset.
*/
typedef struct cb_herald {
    u8      hdr;
    u8      offset;
    ushort  shio_len;
} CBH,*CBHP;
```

```
/*
    Values for CB commands.
*/
#define CB_HERALD           0xcb
#define CB_MAX_DNLD_SIZE    0xd0            /* bsd/sun only allow 256 data in ioctl  */
#define CB_BOARD_LOCATION   0x80002000
#define CB_BOOT_OFFSET      1
#define MAKE_LONG(a,b,c,d) (((u32)a << 24) | ((u32)b << 16) | ((u32)c << 8) | \
                            (u32)d)

/*
    Values for the cmd field of a cb command.
*/
#define CB_BOOT     MAKE_LONG('B','O','O','T')
#define CB_FILL     MAKE_LONG('F','I','L','L')

/* Define some data types for the CB_POKE command */
#define CB_BYTE     1
#define CB_WORD     2
#define CB_LONG     4

/*
    CB responds to a command with the following value.
*/
#define CB_OK       MAKE_LONG('C','B','O','K')
#define CB_POKEC    MAKE_LONG('P','O','K','E')
#define CB_PEEKC    MAKE_LONG('P','E','E','K')

static unsigned char bitswaptbl[256] = {
        0x00, 0x80, 0x40, 0xc0, 0x20, 0xa0, 0x60, 0xe0,
        0x10, 0x90, 0x50, 0xd0, 0x30, 0xb0, 0x70, 0xf0,
        0x08, 0x88, 0x48, 0xc8, 0x28, 0xa8, 0x68, 0xe8,
        0x18, 0x98, 0x58, 0xd8, 0x38, 0xb8, 0x78, 0xf8,
        0x04, 0x84, 0x44, 0xc4, 0x24, 0xa4, 0x64, 0xe4,
        0x14, 0x94, 0x54, 0xd4, 0x34, 0xb4, 0x74, 0xf4,
        0x0c, 0x8c, 0x4c, 0xcc, 0x2c, 0xac, 0x6c, 0xec,
        0x1c, 0x9c, 0x5c, 0xdc, 0x3c, 0xbc, 0x7c, 0xfc,
        0x02, 0x82, 0x42, 0xc2, 0x22, 0xa2, 0x62, 0xe2,
        0x12, 0x92, 0x52, 0xd2, 0x32, 0xb2, 0x72, 0xf2,
        0x0a, 0x8a, 0x4a, 0xca, 0x2a, 0xaa, 0x6a, 0xea,
        0x1a, 0x9a, 0x5a, 0xda, 0x3a, 0xba, 0x7a, 0xfa,
        0x06, 0x86, 0x46, 0xc6, 0x26, 0xa6, 0x66, 0xe6,
        0x16, 0x96, 0x56, 0xd6, 0x36, 0xb6, 0x76, 0xf6,
        0x0e, 0x8e, 0x4e, 0xce, 0x2e, 0xae, 0x6e, 0xee,
        0x1e, 0x9e, 0x5e, 0xde, 0x3e, 0xbe, 0x7e, 0xfe,
        0x01, 0x81, 0x41, 0xc1, 0x21, 0xa1, 0x61, 0xe1,
        0x11, 0x91, 0x51, 0xd1, 0x31, 0xb1, 0x71, 0xf1,
        0x09, 0x89, 0x49, 0xc9, 0x29, 0xa9, 0x69, 0xe9,
        0x19, 0x99, 0x59, 0xd9, 0x39, 0xb9, 0x79, 0xf9,
        0x05, 0x85, 0x45, 0xc5, 0x25, 0xa5, 0x65, 0xe5,
        0x15, 0x95, 0x55, 0xd5, 0x35, 0xb5, 0x75, 0xf5,
        0x0d, 0x8d, 0x4d, 0xcd, 0x2d, 0xad, 0x6d, 0xed,
        0x1d, 0x9d, 0x5d, 0xdd, 0x3d, 0xbd, 0x7d, 0xfd,
        0x03, 0x83, 0x43, 0xc3, 0x23, 0xa3, 0x63, 0xe3,
        0x13, 0x93, 0x53, 0xd3, 0x33, 0xb3, 0x73, 0xf3,
        0x0b, 0x8b, 0x4b, 0xcb, 0x2b, 0xab, 0x6b, 0xeb,
        0x1b, 0x9b, 0x5b, 0xdb, 0x3b, 0xbb, 0x7b, 0xfb,
        0x07, 0x87, 0x47, 0xc7, 0x27, 0xa7, 0x67, 0xe7,
        0x17, 0x97, 0x57, 0xd7, 0x37, 0xb7, 0x77, 0xf7,
        0x0f, 0x8f, 0x4f, 0xcf, 0x2f, 0xaf, 0x6f, 0xef,
        0x1f, 0x9f, 0x5f, 0xdf, 0x3f, 0xbf, 0x7f, 0xff,
};
```

```
/*
    Here are definitions to assist in adding LLC headers or MAC headers
    on transmits and stripping them on receives.

    Every frame sent on an FDDI network must have a MAC header.
*/
#define PAD_LENGTH   3       /* Fill fc out to a word boundary */
struct  machdr {
    u8  machdr_pad[PAD_LENGTH];
    u8  fc;              /* Frame control field       */
    u8  da[6];           /* Destination address       */
    u8  sa[6];           /* Source address            */
};
#define LLC_FC   0x50        /* Our llc fc field, async and long addr */
/*
    Every LLC frame sent on the network must have an LLC header.  The
    format and contents of this header for TCP/IP are defined in
    RFC1103.
*/
struct  llchdr {
    u8      dsap;            /* dsap field                */
    u8      ssap;            /* ssap field                */
    u8      control;         /* control field             */
    u8      proto_id[3];     /* Protocol id field/org code */
    ushort  etype;           /* Ether type field          */
};
#define FIXED_DSAP   0xaa    /* RFC1103 defines dsap and ssap     */
#define FIXED_SSAP   0xaa    /*      for TCP/IP packets.          */
#define PROTO_ID     0       /* It also defines protocol id       */
#define LLC_CONTROL  3       /* and control field as well         */
/*
    The entire header for a TCP/IP frame looks like the following...
*/
struct  fddihdr {
    struct machdr    mh;
    struct llchdr    lh;
};



/*
 * Common RC support definitions and structures
 */

#define RCC_MAX_COMMANDS 256 /* Max commands per command table */
#define RCC_MAX_CMD_MASK (RCC_MAX_COMMANDS - 1)

#define RC_TABLE_INDEX       0
#define FDDI_TABLE_INDEX     1
#define CSP_TABLE_INDEX      2
#define RCC_COMMAND_COUNT    3


/* The index zero is invalid for a command index. */
#define RCC_RC_CMD_BASE        ((RCC_MAX_COMMANDS * RC_TABLE_INDEX) + 1)
#define RCC_FDDI_CMD_BASE      ((RCC_MAX_COMMANDS * FDDI_TABLE_INDEX) + 1)
#define RCC_CSPSMT_CMD_BASE    ((RCC_MAX_COMMANDS * CSP_TABLE_INDEX) + 1)
```

```
/*
 *  Common RC commands
 */
#define RC_CMD_BASE(x)                  ((x) + RCC_RC_CMD_BASE)
#define RC_CMD_RBASE(x)                 ((x) + RC_CMD_BASE(128))

#define RC_BUS_BURST_SET_DIRECTIVE      RC_CMD_BASE(0)
#define RC_BUS_TIMEOUT_SET_DIRECTIVE    RC_CMD_BASE(1)
#define RC_INFO_REQUEST                 RC_CMD_BASE(2)
#define RC_INTERRUPT_SET_DIRECTIVE      RC_CMD_BASE(3)
#define RC_H2B_CHANNEL_CREATE_REQUEST   RC_CMD_BASE(4)
#define RC_B2H_CHANNEL_CREATE_REQUEST   RC_CMD_BASE(5)
#define RC_H2B_LINK_DIRECTIVE           RC_CMD_BASE(6)
#define RC_B2H_SET_DIRECTIVE            RC_CMD_BASE(7)
#define RC_SEG_MEM_DIRECTIVE            RC_CMD_BASE(8)

#define RC_OPERATOR_MSG_INDICATION      RC_CMD_RBASE(0)
#define RC_ERROR_MSG_INDICATION         RC_CMD_RBASE(1)
#define RC_INFO_RESPONSE                RC_CMD_RBASE(2)
#define RC_H2B_CHANNEL_CREATE_RESPONSE  RC_CMD_RBASE(3)
#define RC_B2H_CHANNEL_CREATE_RESPONSE  RC_CMD_RBASE(4)
#define RC_B2H_LINK_INDICATION          RC_CMD_RBASE(5)

/*
 * This is the header for all RC commands and responses.  For a description of
 * the CCB, see p. ? in the text.
 */
typedef struct ccb_s {
    u32 len;        /* length of command including this header    */
    u32 cmd;        /* Command code                               */
} CCB, * CCBP;

/************************* COMMANDS (GENERIC RC) ****************************/

/*
 * RC_BUS_BURST_SET_DIRECTIVE   See Set DMA Burst directive in RC manual
 * for command description
*/
typedef struct rc_bbs_s {
    CCB ccb;
    u32 burst_value;                /* number of transfers per burst        */
} RC_BBS, * RC_BBSP;

/*
 * RC_BUS_TIMEOUT_SET   See Set Bus Timeout directive in RC manual
 * for command description
*/
typedef struct rc_bts_s {
    CCB ccb;
    u32 bus_timeout;                /* number of milliseconds               */
} RC_BTS, * RC_BTSP;

/*
 * RC_INTERRUPT_SET_DIRECTIVE   See Set Interrupt directive in RC manual
 * for command description
*/
typedef struct rc_ris_s {
    CCB ccb;
    u32 channel_id;                 /* Id of channel to set level/vector    */
    u32 level;                      /* Interrupt level                      */
    u32 vector;                     /* Interrupt vector                     */
} RC_RIS, * RC_RISP;
```

```
/*
 * RC_INFO_REQUEST   See Request Statistics in RC manual
 * for command description
*/
typedef struct rc_info_s {
    CCB ccb;
    u32 response_id;                /* Channel id for response              */
    u32 tag;
} RC_INFO, * RC_INFOP;

/*
 * RC_H2B_CHANNEL_CREATE_REQUEST   See Create Host-to-Board Channel in RC manual
 * for command description
*/
typedef struct rc_h2b_channelc_s {
    CCB ccb;
    u32 response_id;                /* Channel id for response              */
    u32 tag;                        /* tag                                  */
    u32 xferopts;                   /* transfer options word                */
    u32 phyaddr;                    /* physical address of the seg          */
    u32 length;                     /* length of segment                    */
} RC_H2B_CHANNELC, * RC_H2B_CHANNELCP;

/*
 * RC_B2H_CHANNEL_CREATE_REQUEST   See Create Board-to-Host Channel in RC manual
 * for command description
*/
typedef struct rc_b2h_channelc_s {
    CCB ccb;
    u32 response_id;                /* Channel id for response              */
    u32 tag;                        /* tag                                  */
} RC_B2H_CHANNELC, * RC_B2H_CHANNELCP;

/*
 * RC_H2B_LINK_DIRECTIVE   See Link Host-to-Board Segment directive in RC manual
 * for command description
*/
typedef struct rc_h2b_link_s {
    CCB ccb;
    u32 xferopts;                   /* transfer options lower 16 bits       */
    u32 phyaddr;                    /* Physical VME Address                 */
    u32 length;                     /* Length of new segment                */
} RC_H2B_LINK, * RC_H2B_LINKP;

/*
 * RC_SEG_MEM_DIRECTIVE   See Allocate Board-to-Host Segment Memory in RC manual
 * for command description
*/
typedef struct rc_seg_mem_s {
    CCB ccb;
    u32 segment_xferopts;           /* Transfer type for segments           */
    u32 segment_length;             /* length of each segment               */
    RC_SEG  seg[1];                 /* one of more Physical addresses       */
} RC_SEGMEM, * RC_SEGMEMP;

/************************** RESPONSES (GENERIC RC) ****************************/

/*
 * RC_OPERATOR_MSG_INDICATION   See Report Printf Call indication in RC manual
 * for command description
*/
typedef struct rc_mfo_s {
    CCB ccb;
    u8  msg[4];                     /* Actually variable length             */
} RC_MFO, * RC_MFOP;
```

```
/*
 * RC_ERROR_MSG_INDICATION   See Error Message indication in RC manual
 * for command description
*/
typedef struct rc_em_s {
    CCB ccb;
    u32 code;
    CCB err_ccb;
    u8  msg[4];                    /* Actually variable length          */
} RC_EM, * RC_EMP;

/*
 * RC_INFO_RESPONSE   See Report Statistics response in RC manual
 * for command description
*/
typedef struct rc_infor_s {
    CCB ccb;
    u32 tag;
    u8  firmware_id[16];
    u8  release_date[16];
    u32 buffer_ram_size;
    u32 static_ram_size;
} RC_INFOR, * RC_INFORP;

/*
 * RC_H2B_CHANNEL_CREATE_RESPONSE   See Report New Host-to-Board Channel in RC manual
 * for command description
*/
typedef struct rc_h2b_channelcr_s {
    CCB ccb;
    u32 tag;
    u32 channel_id;                /* offset into shortio of new channel  */
} RC_H2B_CHANNELCR, * RC_H2B_CHANNELCRP;

/*
 * RC_B2H_CHANNEL_CREATE_RESPONSE   See Report New Board-to-Host Channel in RC manual
 * for command description
*/
typedef struct rc_b2h_channelcr_s {
    CCB ccb;
    u32 tag;
    u32 channel_id;                /* offset into shortio of new channel  */
    u32 vtag;                      /* virtual address of segment          */
    u32 length;                    /* length of segment                   */
} RC_B2H_CHANNELCR, * RC_B2H_CHANNELCRP;

/*
 *  RC_B2H_LINK_INDICATION   See Report Link to Next Segment in RC manual
 * for command description
*/
typedef struct rc_b2h_link_s {
    CCB ccb;
    u32 vtag;                      /* virtual tag from host               */
    u32 length;                    /* Length of new segment               */
} RC_B2H_LINK, * RC_B2H_LINKP;
```

```
/************************* COMMANDS (4211-SPECIFIC) ************************/

/*
 * FDDI-specific structures and commands.
 */

#define FDDI_CMD_BASE(x)                ((x) + RCC_FDDI_CMD_BASE)
#define FDDI_CMD_RBASE(x)               ((x) + FDDI_CMD_BASE(128))

#define FDDI_TX_REQUEST                 FDDI_CMD_BASE(0)
#define FDDI_TXI_REQUEST                FDDI_CMD_BASE(1)     /* TX immediate */
#define FDDI_RX_RESOURCE_DIRECTIVE      FDDI_CMD_BASE(2)
#define FDDI_STATION_ADDRESS_REQUEST    FDDI_CMD_BASE(3)
#define FDDI_STATION_ADDRESS_DIRECTIVE  FDDI_CMD_BASE(4)
#define FDDI_RING_CONTROL_DIRECTIVE     FDDI_CMD_BASE(5)
#define FDDI_MAINT_CONTROL_DIRECTIVE    FDDI_CMD_BASE(6)
#define FDDI_GET_MIB_REQUEST            FDDI_CMD_BASE(7)
#define FDDI_SET_MIB_DIRECTIVE          FDDI_CMD_BASE(8)
#define FDDI_SET_CAM_ADDRESS_REQUEST    FDDI_CMD_BASE(9)
#define FDDI_CLEAR_CAM_ADDRESS_REQUEST  FDDI_CMD_BASE(10)
#define FDDI_FLUSH_CAM_DIRECTIVE        FDDI_CMD_BASE(11)
#define FDDI_TRACE_DIRECTIVE            FDDI_CMD_BASE(12)

#define FDDI_TX_RESPONSE                FDDI_CMD_RBASE(0)
#define FDDI_RX_INDICATION              FDDI_CMD_RBASE(1)
#define FDDI_STATION_ADDRESS_RESPONSE   FDDI_CMD_RBASE(3)
#define FDDI_RING_STATUS_INDICATION     FDDI_CMD_RBASE(4)
#define FDDI_CAM_RESPONSE               FDDI_CMD_RBASE(6)
#define FDDI_GET_MIB_RESPONSE           FDDI_CMD_RBASE(7)
#define FDDI_EVENT_INDICATION           FDDI_CMD_RBASE(8)

/*
    Returned in the FDDI_TX_RESPONSE as the completion status.
*/
#define FDDI_NORMAL                 0x0
#define FDDI_RING_UP                0x01
#define FDDI_RING_DOWN              0x02
#define FDDI_ALLOCB_OUT_OF_MEMORY   0x03

#define FDDI_STATION_ADDRESS_TEMPORARY  0
#define FDDI_STATION_ADDRESS_PERMANENT  1

typedef char SMT_MAC_ADDRESS[6];

/* Defines for status in FDDI_CAM_RESPONSE */
#define FDDI_CAM_SUCCESS            0       /* CAM request successful       */
#define FDDI_CAM_FULL               1       /* CAM table full               */
#define FDDI_CAM_ADDRESS_PRESENT    2       /* CAM address already preset
                                               when trying to set           */
#define FDDI_CAM_ADDRESS_NOT_PRESENT 3      /* CAM address not present
                                               when trying to clear         */

/* For the FDDI_RING_CONTOL_DIRECTIVE ring control field                    */
#define FDDI_RING_CONNECT           1       /* SMT connect                  */
#define FDDI_RING_DISCONNECT        2       /* SMT disconnect               */


/* For the FDDI_MAINT_CONTROL_DIRECTIVE maint_control field                 */
#define FDDI_SMT_MAINT_ON           1       /* Put the PCM state macine     */
                                            /* into the MAINT stat and      */
                                            /* transmit the line state in   */
                                            /* maint_pcm_state              */
#define FDDI_SMT_MAINT_OFF          2       /* Cause the PCM state          */
                                            /* machine to leave the MAINT   */
                                            /* state and return to the      */
                                            /* OFF state                    */
```

```
/* These defines are to be used in the maint_pcm_state                    */
/* of the FDDI_MAINT_CONTROL_DIRECTIVE                                    */
/*                                                                        */
/*      Line State Values                                                 */
/*      Each line state must correspond to an individual bit in an        */
/*      integer value.                                                    */

#define Noise_Line_State              0x02
#define Master_Line_State             0x04
#define Idle_Line_State               0x08
#define Super_Idle_Line_State         Idle_Line_State
#define Halt_Line_State               0x10
#define Quiet_Line_State              0x20
#define Active_Line_State             0x40
#define Transmit_PHY_Data_Request     0x100

typedef struct fddi_physical_seg {
    u32 physaddr;
    u32 length;
} PSEG, * PSEGP;

typedef struct fddi_tx_s {      /* FDDI_TX_REQUEST                        */
    CCB     ccb;                /* See Transmit Raw Data, p. ?, in text   */
    u32     response_id;
    u32     tag;
    u32     class;
    u32     xferopts;
    u32     frame_length;
    u32     fragment_cnt;
    PSEG    tx_seg[1];
} FDDI_TX, *FDDI_TXP;

typedef struct fddi_txi_s {     /* FDDI_TXI_REQUEST                       */
    CCB     ccb;                /* See Transmit Datagram, p. ?, in text   */
    u32     response_id;
    u32     tag;
    u16     dest_addr[3];
    u16     ethertype;
    u32     class;
    u32     xferopts;
    u32     frame_length;
    u32     fragment_cnt;
    PSEG    tx_seg[1];
} FDDI_TXI, * FDDI_TXIP;

/*
    request class parameter
*/
#define FDDI_TX_CLASS_NORESP    0
#define FDDI_TX_CLASS_RESP      1
#define FDDI_TX_CLASS_SYNC      0
#define FDDI_TX_CLASS_ASYNC     2

typedef struct fddi_rx_s {      /* FDDI_RX_RESOURCE_DIRECTIVE                  */
    CCB ccb;                    /* See Set Up Receive Buffers, p. ?, in text   */
    u32 list_xferopts;
    u32 list_physaddr;
    u32 list_length;
    u32 reserved;               /* balance Rx load option                      */
    u32 buffer_xferopts;
    u32 buffer_length;
} FDDI_RX_RES, * FDDI_RX_RESP;

typedef struct buffer_list_element {
    u32 physaddr;
    u32 tag;
} BUFELM, * BUFELMP;
```

```
typedef struct fddi_get_mac_addr {    /* FDDI_STATION_ADDRESS_REQUEST                */
    CCB ccb;                          /* See Request Station Address, p. ?, in text  */
    u32 response_id;
    u32 tag;
} FDDI_GET_MAC, * FDDI_GET_MACP;

typedef struct fddi_set_mac_addr {    /* FDDI_STATION_ADDRESS_DIRECTIVE              */
    CCB ccb;                          /* See Set Station Address, p. ?, in text  */
    u16 mac_addr[3];
    u16 type;
} FDDI_SET_MAC, * FDDI_SET_MACP;

typedef struct fddi_ring_control_dir {    /* FDDI_RING_CONTROL_DIRECTIVE          */
    CCB     ccb;                          /* See Ring Control, p. ?, in text   */
    u32     ring_control;
} FDDI_RING, *FDDI_RINGP;

typedef struct fddi_maint_control_dir {   /* FDDI_MAINT_CONTROL_DIRECTIVE         */
    CCB     ccb;                          /* See Perform Maintenance, p. ?, in text  */
    u32     maint_control;
    u32     maint_pcm_state;
} FDDI_MAINT, *FDDI_MAINTP;

typedef struct fddi_get_mib_request {     /* FDDI_GET_MIB_REQUEST                */
    CCB  ccb;                             /* See Get MIB Attribute, p. ?, in text  */
    u32  response_id;
    u32 tag;
    u32 attr_name;                        /* attribute name           */
    u32 attr_length;                      /* attribute length         */
    u32 attr_index;                       /* attribute index          */
} FDDI_GET_MIB, *FDDI_GET_MIBP;


typedef struct fddi_set_mib_directive {   /* FDDI_SET_MIB_DIRECTIVE              */
    CCB ccb;                              /* See Set MIB Attribute, p. ?, in text  */
    u32  attr_name;                       /* attribute name                      */
    u32 attr_length;                      /* attribute length                    */
    u32 attr_index;                       /* attribute index                     */
    u32  attribute[1];                    /* variable length mib attribute       */
} FDDI_SET_MIB, *FDDI_SET_MIBP;

typedef struct fddi_cam_set_req {         /* FDDI_SET_CAM_ADDRESS_REQUEST        */
    CCB ccb;                              /* See Set CAM Address, p. ?, in text  */
    u32  response_id;
    u32 tag;
    u16 addr[3];
    u16 reserved;
} FDDI_CAMS, *FDDI_CAMSP;


typedef struct fddi_cam_clear_req {       /* FDDI_CLEAR_CAM_ADDRESS_REQUEST      */
    CCB ccb;                              /* See Clear CAM Address, p. ?, in text  */
    u32  response_id;
    u32 tag;
    u16 addr[3];
    u16 reserved;
} FDDI_CAMC, *FDDI_CAMCP;

typedef struct fddi_flush_dir {           /* FDDI_FLUSH_CAM_DIRECTIVE            */
    CCB ccb;                              /* See Flush CAM Addresses, p. ?, in text  */
} FDDI_CAMF, *FDDI_CAMFP;

typedef struct fddi_trace_directive {     /* FDDI_TRACE_DIRECTIVE */
    CCB ccb;                              /* See SMT Trace, p. ?, in text  */
    u32 trace_control;                    /* control frame and event tracing */
  } FDDI_TRACE, *FDDI_TRACEP;
```

```
/************************ RESPONSES (4211-SPECIFIC) ************************/

typedef struct fddi_txr_s {             /* FDDI_TX_RESPONSE                    */
    CCB ccb;                            /* See Tx Response, p. ?, in text      */
    u32 tag;
    u32 tx_stat;
} FDDI_TXR, *FDDI_TXRP;

typedef struct rx_tag_seg {
    u32 tag;
    u32 length;
} TSEG, * TSEGP;

typedef struct fddi_rxr_s {             /* FDDI_RX_INDICATION                  */
    CCB ccb;                            /* See Rx Frame, p. ?, in text      */
    u16 rx_status;
    u16 rx_frame_status;
    u16 frame_length;
    u16 fragment_cnt;
    TSEG rx_seg[1];
} FDDI_RX, *FDDI_RXP;


typedef struct fddi_mac_response_s {    /* FDDI_STATION_ADDRESS_RESPONSE       */
    CCB ccb;                            /* See Report Station Address, p. ?, in text */
    u32 tag;
    u16 canonical[3];
    u16 non_canonical[3];
} FDDI_MACR, * FDDI_MACRP;

typedef struct fddi_ringstatus_s {
    CCB ccb;                            /* See Ring Status Indication, p. ?, in text */
    u32 status;                         /* FDDI_RING_UP FDDI_RING_DOWN         */
} FDDI_RSR, * FDDI_RSP;

typedef struct fddi_cam_res {           /* FDDI_CAM_RESPONSE                   */
    CCB ccb;                            /* See CAM Address Response, p. ?, in text   */
    u32 tag;
    u32 status;
} FDDI_CAMR, *FDDI_CAMRP;

typedef struct fddi_get_mib_response {  /* FDDI_GET_MIB_RESPONSE               */
    CCB ccb;                            /* See MIB Attribute Response, p. ?, in text */
    u32 response_id;
    u32 tag;
    u32 attr_name;                      /* attribute name                */
    u32 attr_length;                    /* attribute length              */
    u32 attr_index;                     /* attribute index               */
    u32 attribute[1];                   /* variable length mib attribute */
} FDDI_GET_MIBR, *FDDI_GET_MIBRP;


typedef struct fddi_event_indication {  /* FDDI_EVENT_INDICATION               */
    CCB ccb;                            /* See SMT Event Indication, p. ?, in text   */
    u32 msg_type;                       /* SMT defined message type            */
    u32 msg[1];                         /* variable length SMT defined event message */
} FDDI_EVENT_R, *FDDI_EVENT_RP;
```

```
/************************************************************************
    SMT Header File

    SMT Definitions Header File
    THIS FILE IS TO BE MODIFIED BY THE IMPLEMENTOR AS NEEDED.

/************************************************************************
    Type Definitions
************************************************************************/
/*
*   Integer Definitions
*   The target environment must be able to support 16- and 32-bit
*   signed and unsigned integer values.
*/
typedef i16     Int16;
typedef u16     uInt16;

typedef i32     Int32;
typedef u32     uInt32;

typedef u32     uInt;


/*
*   Character Definitions
*   The C type "char" is assumed to be a signed 8-bit value.
*/
typedef u8  uChar;

/*
*   Boolean Definitions
*   A Boolean variable may take on one of two values. The FDDI SMT
*   uses the term "flag" for Boolean variables. A flag may either be
*   set or cleared. The constant SET is equivalent to a logical
*   TRUE and the constant CLEAR is a logical FALSE.
*/
typedef int Flag;


/************************************************************************
    System Definitions
************************************************************************/

/*
*   Station Limitations .
*   These values determine the system limits for various elements
*   within the station.
*/

#define       MAX_PATH_COUNT          3
#define       MAX_PORT_COUNT          2
#define       MAX_ATTACH_COUNT        2
#define       MAX_PHY_COUNT           MAX_PORT_COUNT
#define       MAX_MAC_COUNT           2


/*
*   Line State Values
*   Each line state must correspond to an individual bit in an
*   integer value.
*/
#define Noise_Line_State            0x02
#define Master_Line_State           0x04
#define Idle_Line_State             0x08
#define Super_Idle_Line_State       Idle_Line_State
#define Halt_Line_State             0x10
#define Quiet_Line_State            0x20
```

```
#define Active_Line_State           0x40
#define Transmit_PHY_Data_Request   0x100

/*
*   E, A, and C indicator bit positions.
*   The frame interface routines must provide the E, A, and C
*   indicator values with all frames received. The indicator
*   values will be passed to the frame services process as a
*   bit string. These defined values determine the bit positions
*   of each indicator.
*/
#define E_Bit_Position  0x80
#define A_Bit_Position  0x40
#define C_Bit_Position  0x20

/*
*   Losing bid T_Req value.
*   This value is used for Change_Actions in RMT where the MAC is
*   configured to lose the claim process during duplicate address
*   detections.
*/
#define Jam_T_Req_Value ((uInt32) 0x01)



/*****************************************************************************
*   Operational Limitations
*   These value determine the limits of the SMT software.
*****************************************************************************/

/*
*   Signal queue limitations
*/
#define MAX_SIG_QUEUE_SIZE   32

/*
*   FBM Action queue limitations
*/
#define MAX_FBM_QUEUE_SIZE   32

/*
*   Timer list limitations
*/
#define MAX_TIMER_LIST_SIZE     16
#define TICKS_PER_USECS         1
#ifdef  SHORT_TIMES
#define MIN_USECS               ((uTime) 0)
#else
#define MIN_USECS               ((uTime) 1000)
#endif

/*****************************************************************************
*   System Configurations
*   These values describe the target environment for the SMT software.
*****************************************************************************/

/*
*   The bit format used by the processor must be known.
*   Either BIG_ENDIAN or LITTLE_ENDIAN must be defined.
*   Do not define both.
*/
/* #define  LITTLE_ENDIAN */
#define BIG_ENDIAN

#define WORDSIZE    4
```

```
/*
*   Padding sizes.
*   The padding size moves the first character of the frame buffer
*   into the last byte of a word. This first character of the frame
*   buffer is used by the FC field which is usually handled
*   separately from the rest of the frame. By forcing this type of
*   alignment, the remaining data in the frame starts on a word
*   boundary, thus making data transfers easier and more efficient.
*/

#define FC_PAD_SIZE 3


/*****************************************************************************
    DEBUGGING TOOLS
*****************************************************************************/

#ifdef DEBUG
#define SMTDEBUG(level, strng)\
    {   extern  char    *outstring[];\
        if (level) {\
            sprintf strng;\
            DisplayString (level); }\
    }
#else
#define SMTDEBUG(level, strng)
#endif
#ifndef NULL
#define NULL    0
#endif
/*
*   This structure gives access to the various subparts of long words.
*/
union Splice {
    uInt32  l;
    struct {
        uInt16  msw;
        uInt16  lsw;
    } i;
    struct {
        char    b1, b2, b3, b4;
    } c;
};

/*****************************************************************************
       CSP Defined Values
*****************************************************************************/

/*
*       Station Types
*/
#define SMSAS_TYPE      1
#define SMDAS_TYPE      2
#define DMDAS_TYPE      3

/*****************************************************************************
       Interrupt Interface Values
*****************************************************************************/

/*      Interrupt signals */
#define INTFLAG_PHY             0x01
#define INTFLAG_MAC             0x02
#define INTFLAG_FRAME           0x04
#define INTFLAG_MIB             0x08
#define INTFLAG_CSP_TIMER       0x10
#define INTFLAG_FSP_TIMER       0x20
```

```
#define INTFLAG_CSP_MSG         0x40
#define INTFLAG_FSP_MSG         0x80
#define INTFLAG_MSP_MSG         0x100

#define CSP_INTERRUPTS          (INTFLAG_PHY | INTFLAG_MAC\
                                        | INTFLAG_CSP_TIMER | INTFLAG_CSP_MSG)
#define FSP_INTERRUPTS          (INTFLAG_FSP_TIMER | INTFLAG_FSP_MSG)
#define MSP_INTERRUPTS          (INTFLAG_MIB | INTFLAG_MSP_MSG)


/*********************************************************************
    Station Management Header File

    SMT Type Definition Header File

    File:       smttypes.h

    This is the header file used by all SMT modules. This file contains
    type definitions for used by all SMT modules. The type definitions
    supplied here give the SMT system its portability.

/*********************************************************************
    Type Definitions
*********************************************************************/

/*
*   Time Values
*   The CSP system uses two types of time values, one for microseconds
*   and one for seconds.
*/
typedef uInt32  uTime;


/*********************************************************************
*   FDDI Types
*   Some FDDI values exceed the size of the average processor word
*   size. Therefore, structures need to be defined to handle these
*   oversized values.
*********************************************************************/

/*
*   FDDI Addresses
*/
typedef uChar   MACAddr48[6];
typedef uChar   MACAddr16[2];

/*
*   Line States
*/
typedef uInt16  LineState;
```

```
/*************************************************************************
*
*   The following typedefs provide data types and structures
*   for use throughout the MIB. These types align short data
*   values to 32-bit boundaries. This allows the MIB routines
*   to provide attributes properly aligned for frame transmission.
*   Thus, the frame processing routines do not need to know about
*   individual data sizes when building frame parameters.
*
*************************************************************************/

/*
*   8-bit attribute values.
*/
typedef struct SMT8BitStruct {
    uChar       pad[3];
    uChar       data;
} TLV8BitType;

/*
*   16-bit attribute values.
*/
typedef struct SMT16BitStruct {
    uChar       pad[2];
    uInt16      data;
} TLV16BitType;

/*
*   32-bit attribute values.
*   This type allows access to 32-bit attribute using member data
*   as with the 16- and 8-bit attributes.
*/
typedef struct SMT32BitStruct {
    uInt32      data;
} TLV32BitType;

/*
*   Address/ID attribute values.
*   This type allows access to two-byte data field followed by
*   a 6-byte address.
*/
typedef struct SMTIdStruct {
    uChar       data[2];
    MACAddr48   addr;
} TLVIdType;


/*************************************************************************
*
*   Typedefs for special attributes. These attributes are either
*   larger than 32 bits or consists of aggregate members.
*
*************************************************************************/

typedef struct SMTStationIdStruct {
    uChar       user[2];            /* user defined octets      */
    MACAddr48   IEEEAddress;        /* IEEE-assigned address    */
} SMTStationIdType;

typedef struct SMTManufacturerDataStruct {
    uChar       OUI[3];         /* implementor's OUI */
    uChar       data[29];       /* implementor's data */
} SMTManufacturerDataType;

typedef uChar   SMTUserDataType[32];        /* implementor's data */
```

```
typedef struct  SMTTimeStampStruct {
    uInt32      hiword;         /* upper 32 bits */
    uInt32      loword;         /* lower 32 bits */
} SMTTimeStamp;

typedef struct  SMTSetCountStruct {
    uInt32          count;              /* last counter value */
    SMTTimeStamp    setTimeStamp;       /* last time of set */
} SetCountType;

typedef struct  PathConfigStruct {
    uInt16      ResourceType;       /* 2=MAC, 4=PORT */
    uInt16      ResourceIndex;
} PathConfigType;

typedef struct  TLVHdrStruct {
    uInt16  paramType;
    uInt16  paramLen;
} TLVHdrType;

typedef struct  TLVParamStruct {
    uInt16      paramType;      /* Type and length are */
    uInt16      paramLen;       /* common to all params */
    union {
        struct {            /* Some attrs require index */
            TLV16BitType         paramIndex;
            union {
                TLV8BitType     p8;
                TLV16BitType    p16;
                TLV32BitType    p32;
                TLVIdType       pId;
                uChar           pOther[12];
            } pval;
        } otherAttr;
        union {
            TLV8BitType     smt8;   /* SMT attrs need no index */
            TLV16BitType    smt16;
            TLV32BitType    smt32;
            TLVIdType       smtId;
            uChar           smtOther[32];
        } smtAttr;
    } paramValue;
} TLVParamType;

typedef struct  TLVSetCountStruct {
    uInt16          paramType;
    uInt16          paramLen;
    SetCountType    paramValue;
} TLVSetCountType;

/*********************************************************************
*
*   The following typedefs are left of backward compatibility to
*   older versions of the XLNT Manager software.
*
*********************************************************************/

/*
*   FDDI Timer type.
*/
typedef uInt32 FDDITimer;

/*
*   The data type for manufacturer data fields.
*/
typedef SMTManufacturerDataType  SMTManufData;
```

```
/*
*   SMT Station id.  This value is recommended to be set to
*   the MAC address of the first MAC in the station.
*/
typedef SMTStationIdType SMTStationId;

/*
*   SMT User Data field type.
*/
typedef SMTUserDataType  SMTUserData;

/*
*   The list of supported SMT versions.  The supported versions
*   must be a contigous range.  The structure below gives the
*   starting version supported and the ending version supported.
*/
typedef struct SMTVersList {
    uInt16  Start;          /* Initial version supported */
    uInt16  End;            /* Last version supported */
} SMTVersList;


/*****************************************************************************
*
*   Typedefs for events and conditions as defined in the MIB.
*
*****************************************************************************/

/* Configuration Change Event */
typedef struct EvtCfgChgStruct {
    TLVHdrType  eventHdr;       /* event header */
    TLVHdrType  CF_StateHdr;    /* CF_State header */
    TLV8BitType CF_State;       /* CF_State value */
#ifdef OPTIONAL_PARAMETER
    TLVHdrType  PathListHdr;    /* Path list header */
    TLV16BitType    pathIndex;  /* index for path */
                                /* maximum size path list */
    uChar       PathList[(MAX_MAC_COUNT + MAX_PORT_COUNT) * 4];
#endif
}
EvtCfgChgType;

/* Duplicate Address Condition */
typedef struct CondDAStruct {
    TLVHdrType  condHdr;            /* condition header */
    uInt16      Condition;          /* condition status */
    uInt16      MACIndex;           /* index (MIB value) */
    MACAddr48   DuplicateAddr;      /* this duplicate */
    MACAddr48   UNADuplicateAddr;   /* UNA duplicate */
} CondDAType;

/* Frame Error Condition */
typedef struct CondFrErrStruct {
    TLVHdrType      condHdr;    /* condition header */
    uInt16          Condition_State;
    uInt16          MACIndex;   /* index (MIB value) */
    uInt32          Frame_Ct;
    uInt32          Error_Ct;
    uInt32          Lost_Ct;
    uInt32          BaseFrame_Ct;
    uInt32          BaseError_Ct;
    uInt32          BaseLost_Ct;
    SMTTimeStamp    BaseTimeStamp;
    uChar           pad[2];
    uInt16          FrameErrorRatio;
} CondFrErrType;
```

```
/* Not Copied Condition */
typedef struct CondNotCopiedStruct {
    TLVHdrType      condHdr;    /* condition header */
    uInt16          Condition_State;
    uInt16          MACIndex;   /* index (MIB value) */
    uInt32          NotCopied_Ct;
    uInt32          Copied_Ct;
    uInt32          BaseNotCopied_Ct;
    SMTTimeStamp    BaseTimeStamp;
    uInt32          BaseCopied_Ct;
    uChar           pad[2];
    uInt16          NotCopiedRatio;
} CondNotCopiedType;


/* MAC Neighbor Change Event */
typedef struct EvtNbrChgStruct {
    TLVHdrType  eventHdr;
    uInt16      Condition;
    uInt16      MACIndex;
    MACAddr48   Old_UNA;
    MACAddr48   New_UNA;
    MACAddr48   Old_DNA;
    MACAddr48   New_DNA;
} EvtNbrChgType;

/* Trace Status Event */
typedef struct EvtTrStatType {
    TLVHdrType  eventHdr;
    uChar       pad;
    uChar       TraceStarted;
    uChar       TraceTerminated;
    uChar       TracePropagated;
} EvtTrStatType;

/* Ler Condition */
typedef struct CondLerStruct {
    TLVHdrType      condHdr;
    uInt16          ConditionState;
    uInt16          PORTIndex;
    uChar           pad1;
    uChar           Ler_Cutoff;
    uChar           Ler_Alarm;
    uChar           Ler_Estimate;
    uInt32          Lem_Reject_Ct;
    uInt32          Lem_Ct;
    uChar           pad2[3];
    uChar           BaseLer_Estimate;
    uInt32          BaseLem_Reject_Ct;
    uInt32          BaseLem_Ct;
    SMTTimeStamp    BaseLer_TimeStamp;
} CondLerType;

/* Undesired Connection Attempt Event */
typedef struct EvtConnectStruct {
    TLVHdrType  eventHdr;
    uChar       pad[2];
    uInt16      PORTIndex;
    uChar       PC_Type;
    uChar       connectState;
    uChar       PC_Neighbor;
    uChar       connectionAccepted;
} EvtConnectType;
```

```
/* EB Error Condition */
typedef struct CondEBErrStruct {
    TLVHdrType   condHdr;
    uInt16       ConditionState;
    uInt16       PORTIndex;
    uInt32       EbError_Ct;
} CondEBErrType;


/*****************************************************************************
*   Constant Definitions
*****************************************************************************/

/*
*   Boolean values
*/

#ifndef FALSE
#define FALSE   0
#endif

#ifndef TRUE
#define TRUE    !FALSE
#endif

#ifndef CLEAR
#define CLEAR   FALSE
#endif

#ifndef SET
#define SET     1
#endif

#ifndef NULL
#define NULL    0
#endif

#define DISABLE     FALSE
#define ENABLE      TRUE

#define INSERTED    TRUE
#define DEINSERTED  FALSE


/*****************************************************************************
*   Macro Definitions
*****************************************************************************/

/*
*   Definitions to access parameter values.
*
*   Use the macros to access the appropriate parameter element
*   in the TLVParamType. To use them, put the element reference
*   in front of the macro as follows:
*       x.SMTPARAM8
*       y->MACINDEX
*   where x is of type TLVParamType and y is a pointer to a
*   TLVParamType.
*/
#define SMTPVAL         paramValue.smtAttr
#define SMT_PARAM8      paramValue.smtAttr.smt8.data
#define SMTPARAM16      paramValue.smtAttr.smt16.data
#define SMTPARAM32      paramValue.smtAttr.smt32.data
#define SMT_ADDR        paramValue.smtAttr.smtId.addr
#define SMTOTHER        paramValue.smtAttr.smtOther
```

```
#define MACPVAL          paramValue.otherAttr.pval
#define MACINDEX         paramValue.otherAttr.paramIndex.data
#define MACPARAM8        paramValue.otherAttr.pval.p8.data
#define MACPARAM16       paramValue.otherAttr.pval.p16.data
#define MACPARAM32       paramValue.otherAttr.pval.p32.data
#define MACADDR          paramValue.otherAttr.pval.pId.addr
#define MACOTHER         paramValue.otherAttr.pval.pOther


#define PATHPVAL         paramValue.otherAttr.pval
#define PATHINDEX        paramValue.otherAttr.paramIndex.data
#define PATHPARAM8       paramValue.otherAttr.pval.p8.data
#define PATHPARAM16      paramValue.otherAttr.pval.p16.data
#define PATHPARAM32      paramValue.otherAttr.pval.p32.data
#define PATHOTHER        paramValue.otherAttr.pval.pOther

#define PORTPVAL         paramValue.otherAttr.pval
#define PORTINDEX        paramValue.otherAttr.paramIndex.data
#define PORTPARAM8       paramValue.otherAttr.pval.p8.data
#define PORTPARAM16      paramValue.otherAttr.pval.p16.data
#define PORTPARAM32      paramValue.otherAttr.pval.p32.data
#define PORTOTHER        paramValue.otherAttr.pval.pOther

#define ATTACHPVAL       paramValue.otherAttr.pval
#define ATTACHINDEX      paramValue.otherAttr.paramIndex.data
#define ATTACHPARAM8     paramValue.otherAttr.pval.p8.data
#define ATTACHPARAM16    paramValue.otherAttr.pval.p16.data
#define ATTACHPARAM32    paramValue.otherAttr.pval.p32.data
#define ATTACHOTHER      paramValue.otherAttr.pval.pOther


/*********************************************************************
    Management Information Base Header File

    MIB attribute and structure definition.

    File:       mibdefs.h

    This file contains types and constants required for accessing
    the Management Information Base (MIB).  The MIB is accessed thru
    get, set, add and change routines.  It contains or has access to all
    SMT attributes within a particular station.

    Modification History:

    *** CREATED FOR SMT 6.2 ***

*********************************************************************/

/*********************************************************************
*
*   The following constants are used to access particular station
*   attributes.  The attributes maintained on a station are defined
*   in section 3.4 of the X3T9.5 FDDI Station Management
*   Specification.  When reading or updating an attribute the
*   particular attribute being read/updated must be identified.
*   The following constants perform this role.
*
*********************************************************************/
```

```
/*
*   The SMT version that is currently running on the machine.  All
*   versions up to this point are known as version 1.  This parameter
*   should not be arbitrarily changed.  SMTs running on other vendors
*   equipment expect certain characteristics depending on the version
*   of SMT which is being run.  The current implementation only knows
*   characteristics about SMT 6.2.
*/
#define SMTOPVERSION 0001
#define SMTHIVERSION 0001
#define SMTLOVERSION 0001


/************************************************************************
    The attribute ID values correspond to the parameter type
    values listed in the SMT documentation. The high byte contains
    the object and sub-object ID, and the lower byte is the
    attribute registration ID.

    The naming convientions used here is based on the attribute
    names used in the SMT standard. In most cases, the constant
    exactly matches the name used in the standard. In a few cases,
    minor alterations were made for readability and syntax. The
    dash (-) used by the standard is converted to an underscore
    (_) for C.
************************************************************************/

/*
*   Object ID constants.
*/
#define fddiSMT             0x1000
#define fddiMAC             0x2000
#define fddiPATHClass       0x3000
#define fddiPATHClassPATH   0x3200
#define fddiPORT            0x4000
#define fddiATTACHMENT      0x6000

/*
*   ID Masks.
*/
#define MIB_OBJECT_MASK     0xF000
#define MIB_SUBOBJECT_MASK  0x0F00
#define MIB_ID_MASK         0x00FF

/*
*   SMT Attribute Constants
*/
#define fddiSMTStationIdGrp         (fddiSMT | 10)
#define fddiSMTStationId            (fddiSMT | 11)
#define fddiSMTOpVersionId          (fddiSMT | 13)
#define fddiSMTHiVersionId          (fddiSMT | 14)
#define fddiSMTLoVersionId          (fddiSMT | 15)
#define fddiSMTManufacturerData     (fddiSMT | 16)
#define fddiSMTUserData             (fddiSMT | 17)

#define fddiSMTStationConfigGrp     (fddiSMT | 20)
#define fddiSMTMAC_Ct               (fddiSMT | 21)
#define fddiSMTNonMaster_Ct         (fddiSMT | 22)
#define fddiSMTMaster_Ct            (fddiSMT | 23)
#define fddiSMTPathsAvailable       (fddiSMT | 24)
#define fddiSMTConfigCapabilities   (fddiSMT | 25)
#define fddiSMTConfigPolicy         (fddiSMT | 26)
#define fddiSMTConnectionPolicy     (fddiSMT | 27)
#define fddiSMTReportLimit          (fddiSMT | 28)
#define fddiSMTT_Notify             (fddiSMT | 29)
#define fddiSMTStatusReporting      (fddiSMT | 30)
```

```
#define fddiSMTStatusGrp              (fddiSMT | 40)
#define fddiSMTECMState               (fddiSMT | 41)
#define fddiSMTCF_State               (fddiSMT | 42)
#define fddiSMTHoldState              (fddiSMT | 43)
#define fddiSMTRemoteDisconnectFlag   (fddiSMT | 44)

#define fddiSMTMIBOperationGrp        (fddiSMT | 50)
#define fddiSMTMsgTimeStamp           (fddiSMT | 51)
#define fddiSMTTransitionTimeStamp    (fddiSMT | 52)
#define fddiSMTSetCount               (fddiSMT | 53)
#define fddiSMTLastSetStationId       (fddiSMT | 54)

/* XDI added attributes */
#define xdiSMTBothWrapCapability      (fddiSMT | 201)
#define xdiSMTBothWrapPolicy          (fddiSMT | 202)
#define xdiSMTStationType             (fddiSMT | 203)
#define xdiSMTPathLatencyRing1        (fddiSMT | 204)
#define xdiSMTPathLatencyRing2        (fddiSMT | 205)
#define xdiSMTTopology                (fddiSMT | 206)
#define xdiSMTOutIndex1               (fddiSMT | 207)
#define xdiSMTOutIndex2               (fddiSMT | 208)
#define xdiSMTSB_Flag                 (fddiSMT | 209)

/*
*   MAC Attribute Constants
*/
#define fddiMACCapabilitiesGrp          (fddiMAC | 10)  /* READ-ONLY */
#define fddiMACFrameStatusCapabilities  (fddiMAC | 11)  /* READ-ONLY */
#define fddiMACBridgeFunction           (fddiMAC | 12)  /* READ-ONLY */
#define fddiMACT_MaxGreatestLowerBound  (fddiMAC | 13)  /* READ-ONLY */
#define fddiMACTVXGreatestLowerBound    (fddiMAC | 14)  /* READ-ONLY */

#define fddiMACConfigGrp                (fddiMAC | 20)  /* READ-ONLY */
#define fddiMACPathsAvailable           (fddiMAC | 22)  /* READ-ONLY */
#define fddiMACCurrentPath              (fddiMAC | 23)  /* READ-ONLY */
#define fddiMACUpstreamNbr              (fddiMAC | 24)  /* READ-ONLY */
#define fddiMACDownstreamNbr            (fddiMAC | 25)  /* READ-ONLY */
#define fddiMACOldUpstreamNbr           (fddiMAC | 26)  /* READ-ONLY */
#define fddiMACOldDownstreamNbr         (fddiMAC | 27)  /* READ-ONLY */
#define fddiMACRootConcentratorMAC      (fddiMAC | 28)  /* READ-ONLY */
#define fddiMACDup_Addr_Test            (fddiMAC | 29)  /* READ-ONLY */
#define fddiMACPathsRequested           (fddiMAC | 32)
#define fddiMACDownstreamPORTType       (fddiMAC | 33)  /* READ-ONLY */

#define fddiMACAddressGrp               (fddiMAC | 40)  /* READ-ONLY */
#define fddiMACSMTAddress               (fddiMAC | 41)
#define fddiMACLongAliases              (fddiMAC | 42)
#define fddiMACShortAliases             (fddiMAC | 43)
#define fddiMACLongGrpAddrs             (fddiMAC | 44)
#define fddiMACShortGrpAddrs            (fddiMAC | 45)

#define fddiMACOperationGrp             (fddiMAC | 50)  /* READ-ONLY */
#define fddiMACT_Req                    (fddiMAC | 51)
#define fddiMACT_Neg                    (fddiMAC | 52)  /* READ-ONLY */
#define fddiMACT_Max                    (fddiMAC | 53)  /* READ-ONLY */
#define fddiMACTvxValue                 (fddiMAC | 54)  /* READ-ONLY */
#define fddiMACT_Min                    (fddiMAC | 55)
#define fddiMACT_Pri0                   (fddiMAC | 56)  /* READ-ONLY */
#define fddiMACT_Pri1                   (fddiMAC | 57)  /* READ-ONLY */
#define fddiMACT_Pri2                   (fddiMAC | 58)  /* READ-ONLY */
#define fddiMACT_Pri3                   (fddiMAC | 59)  /* READ-ONLY */
#define fddiMACT_Pri4                   (fddiMAC | 60)  /* READ-ONLY */
#define fddiMACT_Pri5                   (fddiMAC | 61)  /* READ-ONLY */
#define fddiMACT_Pri6                   (fddiMAC | 62)  /* READ-ONLY */
#define fddiMACFrameStatus              (fddiMAC | 63)

#define fddiMACCountersGrp              (fddiMAC | 70)  /* READ-ONLY */
```

```
#define fddiMACFrame_Ct                    (fddiMAC | 71)  /* READ-ONLY */
#define fddiMACCopied_Ct                   (fddiMAC | 72)  /* READ-ONLY */
#define fddiMACTransmit_Ct                 (fddiMAC | 73)  /* READ-ONLY */
#define fddiMACToken_Ct                    (fddiMAC | 74)  /* READ-ONLY */

#define fddiMACErrorCtrsGrp                (fddiMAC | 80)  /* READ-ONLY */
#define fddiMACError_Ct                    (fddiMAC | 81)  /* READ-ONLY */
#define fddiMACLost_Ct                     (fddiMAC | 82)  /* READ-ONLY */
#define fddiMACTvxExpired_Ct               (fddiMAC | 83)  /* READ-ONLY */
#define fddiMACNotCopied_Ct                (fddiMAC | 84)  /* READ-ONLY */
#define fddiMACLate_Ct                     (fddiMAC | 85)  /* READ-ONLY */
#define fddiMACRingOp_Ct                   (fddiMAC | 86)  /* READ-ONLY */

#define fddiMACFrameErrorConditionGrp      (fddiMAC | 90)  /* READ-ONLY */
#define fddiMACBaseFrame_Ct                (fddiMAC | 91)  /* READ-ONLY */
#define fddiMACBaseError_Ct                (fddiMAC | 92)  /* READ-ONLY */
#define fddiMACBaseLost_Ct                 (fddiMAC | 93)  /* READ-ONLY */
#define fddiMACBaseTimeFrameError          (fddiMAC | 94)  /* READ-ONLY */
#define fddiMACFrameErrorThreshold         (fddiMAC | 95)
#define fddiMACFrameErrorRatio             (fddiMAC | 96)  /* READ-ONLY */

#define fddiMACNotCopiedConditionGrp       (fddiMAC | 100) /* READ-ONLY */
#define fddiMACBaseNotCopied_Ct            (fddiMAC | 101) /* READ-ONLY */
#define fddiMACBaseTimeNotCopied           (fddiMAC | 102) /* READ-ONLY */
#define fddiMACNotCopiedThreshold          (fddiMAC | 103)
#define fddiMACBaseCopied_Ct               (fddiMAC | 104) /* READ-ONLY */
#define fddiMACNotCopiedRatio              (fddiMAC | 105) /* READ-ONLY */

#define fddiMACStatusGrp                   (fddiMAC | 110) /* READ-ONLY */
#define fddiMACRMTState                    (fddiMAC | 111) /* READ-ONLY */
#define fddiMACDa_Flag                     (fddiMAC | 112) /* READ-ONLY */
#define fddiMACUnaDa_Flag                  (fddiMAC | 113) /* READ-ONLY */
#define fddiMACFrameErrorCondition         (fddiMAC | 114) /* READ-ONLY */
#define fddiMACNotCopiedCondition          (fddiMAC | 115) /* READ-ONLY */
#define fddiMACLLCServiceAvailable         (fddiMAC | 116) /* READ-ONLY */

#define fddiMACRootMACStatusGrp            (fddiMAC | 120) /* READ-ONLY */
#define fddiMACMasterSlaveLoopStatus       (fddiMAC | 121) /* READ-ONLY */
#define fddiMACRootMACDownstreamPORTType   (fddiMAC | 122) /* READ-ONLY */
#define fddiMACRootMACCurrentPath          (fddiMAC | 123) /* READ-ONLY */

/* XDI added attributes */
#define xdiMACRMTDupPolicy1                (fddiMAC | 201)
#define xdiMACRMTDupPolicy2                (fddiMAC | 202)
#define xdiMACResourceIndex                (fddiMAC | 203) /* READ-ONLY */
#define xdiMACConnectedIndex               (fddiMAC | 204) /* READ-ONLY */
#define xdiMACDataAddress                  (fddiMAC | 205)
#define xdiMACOperational                  (fddiMAC | 207)

/*
*   Path Class Attributes
*/
#define fddiPATHConfigGrp                      (fddiPATHClass | 10)
#define fddiPATHClassTrace_MaxExpiration       (fddiPATHClass | 12)
#define fddiPATHClassTVXLowerBound             (fddiPATHClass | 13)
#define fddiPATHClassT_MaxLowerBound           (fddiPATHClass | 14)

/*
*   Path Class Path Attributes
*/
#define fddiPATHClassPATHConfigGrp         (fddiPATHClassPATH | 10)    /* READ-ONLY */
#define fddiPATHClassPATHClassType         (fddiPATHClassPATH | 11)    /* READ-ONLY */
#define fddiPATHClassPATHPORTOrder         (fddiPATHClassPATH | 12)    /* READ-ONLY */
#define fddiPATHClassPATHRingLatency       (fddiPATHClassPATH | 13)
#define fddiPATHClassPATHTraceStatus       (fddiPATHClassPATH | 14)    /* READ-ONLY */
#define fddiPATHClassPATHSba               (fddiPATHClassPATH | 15)
#define fddiPATHClassPATHSbaOverhead       (fddiPATHClassPATH | 16)
```

```
#define fddiPATHClassPATHStatus          (fddiPATHClassPATH | 17)     /* READ-ONLY */
#define fddiPATHClassPATHConfiguration   (fddiPATHClassPATH | 18)     /* READ-ONLY */
#define fddiPATHClassPATHT_Rmode         (fddiPATHClassPATH | 19)

/*
*   Port Attributes
*/
#define fddiPORTConfigGrp                (fddiPORT | 10)              /* READ-ONLY */
#define fddiPORTPC_Type                  (fddiPORT | 12)              /* READ-ONLY */
#define fddiPORTPC_Neighbor              (fddiPORT | 13)              /* READ-ONLY */
#define fddiPORTConnectionPolicies       (fddiPORT | 14)
#define fddiPORTRemoteMACIndicated       (fddiPORT | 15)              /* READ-ONLY */
#define fddiPORTCE_State                 (fddiPORT | 16)              /* READ-ONLY */
#define fddiPORTPathsRequested           (fddiPORT | 17)
#define fddiPORTMACPlacement             (fddiPORT | 18)              /* READ-ONLY */
#define fddiPORTAvailablePaths           (fddiPORT | 19)              /* READ-ONLY */
#define fddiPORTMACLoop_Time             (fddiPORT | 21)
#define fddiPORTFotxClass                (fddiPORT | 22)              /* READ-ONLY */

#define fddiPORTOperationGrp             (fddiPORT | 30)              /* READ-ONLY */
#define fddiPORTMaintLineState           (fddiPORT | 31)
#define fddiPORTTB_Max                   (fddiPORT | 32)
#define fddiPORTBS_Flag                  (fddiPORT | 33)              /* READ-ONLY */

#define fddiPORTErrorCtrsGrp             (fddiPORT | 40)              /* READ-ONLY */
#define fddiPORTEBError_Ct               (fddiPORT | 41)              /* READ-ONLY */
#define fddiPORTLCTFail_Ct               (fddiPORT | 42)              /* READ-ONLY */

#define fddiPORTLerGrp                   (fddiPORT | 50)              /* READ-ONLY */
#define fddiPORTLer_Estimate             (fddiPORT | 51)              /* READ-ONLY */
#define fddiPORTLem_Reject_Ct            (fddiPORT | 52)              /* READ-ONLY */
#define fddiPORTLem_Ct                   (fddiPORT | 53)              /* READ-ONLY */
#define fddiPORTBaseLer_Estimate         (fddiPORT | 54)              /* READ-ONLY */
#define fddiPORTBaseLem_Reject_Ct        (fddiPORT | 55)              /* READ-ONLY */
#define fddiPORTBaseLem_Ct               (fddiPORT | 56)              /* READ-ONLY */
#define fddiPORTBaseLer_TimeStamp        (fddiPORT | 57)              /* READ-ONLY */
#define fddiPORTLer_Cutoff               (fddiPORT | 58)
#define fddiPORTLer_Alarm                (fddiPORT | 59)

#define fddiPORTStatusGrp                (fddiPORT | 60)              /* READ-ONLY */
#define fddiPORTConnectState             (fddiPORT | 61)              /* READ-ONLY */
#define fddiPORTPCMState                 (fddiPORT | 62)              /* READ-ONLY */
#define fddiPORTPC_Withhold              (fddiPORT | 63)              /* READ-ONLY */
#define fddiPORTLerCondition             (fddiPORT | 64)              /* READ-ONLY */

/* XDI added attributes */
#define xdiPORTResourceIndex             (fddiPORT | 201)             /* READ-ONLY */
#define xdiPORTConnectedIndex            (fddiPORT | 202)             /* READ-ONLY */
#define xdiPORTOperational               (fddiPORT | 203)

/*
*   Attachment Attributes
*/
#define fddiATTACHMENTConfigGrp              (fddiATTACHMENT | 10)    /* READ-ONLY */
#define fddiATTACHMENTClass                  (fddiATTACHMENT | 11)    /* READ-ONLY */
#define fddiATTACHMENTOpticalBypassPresent   (fddiATTACHMENT | 12)    /* READ-ONLY */
#define fddiATTACHMENTI_MaxExpiration        (fddiATTACHMENT | 13)    /* READ-ONLY */
#define fddiATTACHMENTInsertedStatus         (fddiATTACHMENT | 14)    /* READ-ONLY */
#define fddiATTACHMENTInsertPolicy           (fddiATTACHMENT | 15)
```

```
/************************************************************************
    MIB Actions
************************************************************************/

#define fddiSMTStationAction          (fddiSMT | 60)

#define fddiMACAction                 (fddiMAC | 130)

#define fddiPORTAction                (fddiPORT | 70)


/************************************************************************
    MIB Events
************************************************************************/

#define fddiSMTConfigurationChgEvent          (fddiSMT | 70)

#define fddiMACDuplicateAddressCondition      (fddiMAC | 140)
#define fddiMACFrameErrorConditionEvent       (fddiMAC | 141)
#define fddiMACNotCopiedConditionEvent        (fddiMAC | 142)
#define fddiMACNeighborChangeEvent            (fddiMAC | 143)

#define fddiPATHTraceStatusEvent              (fddiPATHClassPATH | 30)

#define fddiPORTLerConditionEvent             (fddiPORT | 80)
#define fddiPORTUndesiredConnectionAttempt    (fddiPORT | 81)
#define fddiPORTEBErrorConditionEvent         (fddiPORT | 82)

/************************************************************************
    Station Management Header File

    FDDI Header File

    File:       fddihdr.h
    This is the header file contains all definitions and constant
    values specified in the FDDI SMT documentation.

    Modification History:

    *** Updated to SMT 6.2 ***

************************************************************************/

/************************************************************************
    Timer Values
************************************************************************/

/*
*   These time values are specified in the FDDI SMT Connection Management
*   Facilities documentation. For convenience in handling time values, all
*   times are expressed in microseconds (usecs).
*/

/*  Timer Calculation Elements */
#define D_Max               (uTime) 1773

/*  ECM Expiration Values */
#define I_Max_Default       (uTime) 25000
#define IN_Max              (uTime) 40000
#define TD_Min              (uTime) 5000
#define Trace_Max_Default   (uTime) 7000000
#define Min_Trace_Max       (uTime) 6000        /* min value for Trace_Max */

/* PCM Expiration Values */
#define C_Min               (uTime) 1600
#define TL_Min              (uTime) 30
```

```
/* LS_Min    0.48 usecs must be handled by hardware */
#define TB_Min                (uTime) 5000
#define TB_Max_Default        (uTime) 50000
#define T_Out                 (uTime) 100000
#define NS_Max                (uTime) 1300
#define LC_Short              (uTime) 50000
#define LC_Medium             (uTime) 500000
#define LC_Long               (uTime) 5000000
#define LC_Extended           (uTime) 50000000
#define T_Next_9              (uTime) 200000

/* CFM Expiration Values */
#define T_Scrub               (uTime) 3500

/* RMT Expiration Values */
#define T_Non_OP              (uTime) 1000000
#define T_Stuck               (uTime) 8000000
#define T_Direct              (uTime) 370000
#define T_Jam                 (uTime) 370000
#define T_DBJ                 (uTime) 82000
#define T_Announce            (uTime) 2500000
#define T_Rmode_Default       (uTime) 0
#define Dup_Max               (D_Max + D_Max)
    /*
    * The amount of skew is the amount of time to reduce T_Max
    * for checking the RM(34c) transition.
    */
#define T_RM34c               (uTime) 5000

/* FBM Expiration Values */
/*
*   Note: These values represent time in seconds.
*/
#define T_Notify_Default      30
#define T_NN_Out              228


/*************************************************************************
    CMT & RMT States
*************************************************************************/

/*
*   The following defined values are used to determine the current
*   states for each state machine. The values and names used follow
*   the FDDI SMT documentation.
*/

/*
*   ECM States
*/
#define EC_OUT          0
#define EC_IN           1
#define EC_TRACE        2
#define EC_LEAVE        3
#define EC_PATH_TEST    4
#define EC_INSERT       5
#define EC_CHECK        6
#define EC_DEINSERT     7
```

```
/*
*   CFM States
*/
#define CF_ISOLATED      0
#define CF_WRAP_S        1
#define CF_WRAP_A        2
#define CF_WRAP_B        3
#define CF_WRAP_AB       4
#define CF_THRU          5

/*
*   CCE States
*/
#define CE_ISOLATED      0
#define CE_INSERT_P      1
#define CE_INSERT_S      2
#define CE_INSERT_X      3
#define CE_LOCAL         4

/*
*   PCM States
*/
#define PC_OFF       0
#define PC_BREAK     1
#define PC_TRACE     2
#define PC_CONNECT   3
#define PC_NEXT      4
#define PC_SIGNAL    5
#define PC_JOIN      6
#define PC_VERIFY    7
#define PC_ACTIVE    8
#define PC_MAINT     9

/*
*   RMT States
*/
#define RM_ISOLATED      0
#define RM_NON_OP        1
#define RM_RING_OP       2
#define RM_DETECT        3
#define RM_NON_OP_DUP    4
#define RM_RING_OP_DUP   5
#define RM_DIRECTED      6
#define RM_TRACE         7

/*
*   RMT Substates used in XDI SMT implementation.
*/
#define RM_JAM_A_WAIT_TJAM   41
#define RM_JAM_B_WAIT_TDBJ   42
#define RM_JAM_B_WAIT_TJAM   43

/*
*   LEM States used in XDI SMT implementation.
*/
#define LE_OFF       0
#define LE_MONITOR   1
#define LE_LCT       2
```

```
/************************************************************************
    SMT Definitions
************************************************************************/

/*
*   Configuration Capabilities/Policies
*/
#define Config_None             0x0
#define Config_Hold_Available   0x0001
#define Config_CF_Wrap_AB       0x0002


/*
*   Connection Policies
*/
#define Policy_rejectA_A    0x0001
#define Policy_rejectA_B    0x0002
#define Policy_rejectA_S    0x0004
#define Policy_rejectA_M    0x0008
#define Policy_rejectB_A    0x0010
#define Policy_rejectB_B    0x0020
#define Policy_rejectB_S    0x0040
#define Policy_rejectB_M    0x0080
#define Policy_rejectS_A    0x0100
#define Policy_rejectS_B    0x0200
#define Policy_rejectS_S    0x0400
#define Policy_rejectS_M    0x0800
#define Policy_rejectM_A    0x1000
#define Policy_rejectM_B    0x2000
#define Policy_rejectM_S    0x4000
#define Policy_rejectM_M    0x8000
/* Additional policy types used by XLNT Manager */
#define Policy_limited      (~(Policy_rejectA_B | Policy_rejectB_A\
                    | Policy_rejectS_M | Policy_rejectM_S))
#define Policy_expanded     (Policy_rejectM_M)
#define Policy_acceptAll    0x0
#define Policy_rejectAll    0xFFFF

/*
*   Hold State
*/
#define HS_Not_Holding  0
#define HS_Holding_Prm  1           /* holding on primary */
#define HS_Holding_Sec  2           /* holding on secondary */

/*
*   T_Notify Limits
*/
#define T_Notify_Min        2
#define T_Notify_Max       30

/*
*   Station Actions
*/
#define StationAction_Connect       0
#define StationAction_Disconnect    1
#define StationAction_Path_Test     2
#define StationAction_Self_Test     3
```

```
/*************************************************************************
    MAC Definitions
*************************************************************************/

/*
*   Frame Status Capabilities
*/
#define FSC_None          0
#define FSC_Type0         0x0001
#define FSC_Type1         0x0002
#define FSC_Type2         0x0004
#define FSC_Type0_Pgm     0x0100
#define FSC_Type1_Pgm     0x0200
#define FSC_Type2_Pgm     0x0400


/*
*   Bridge Function
*/
#define Bridge_None       0
#define Bridge_Type0      0x01
#define Bridge_Type1      0x02

/*
*   Master-Slave Loop Status
*/
#define MSLoop_Unknown        0
#define MSLoop_Suspected      1
#define MSLoop_None           2

/*
*   Duplicate Address Test
*/
#define DA_Test_None          0
#define DA_Test_Pass          1
#define DA_Test_Fail          2

/*
*   Neighbor Change Event Conditions
*/
#define EVENT_UNA_CHANGED     0x01
#define EVENT_DNA_CHANGED     0x02

/*
*   MAC Actions
*/
#define MACAction_EnableLLCService    0
#define MACAction_DisableLLCService   1
#define MACAction_ConnectMAC          2
#define MACAction_DisconnectMAC       3


/*************************************************************************
    PATH Definitions
*************************************************************************/

/*
*   Paths Available/Current Path
*/
#define PA_UNKNOWN        0x00
#define PA_PRIMARY        0x01
#define PA_SECONDARY      0x02
#define PA_LOCAL          0x04
#define PA_ISOLATED       0x08
```

```
/*
*   PATH Order
*/
#define Path_Order_Unsupported   0
#define Path_Order_Ascending     1
#define Path_Order_Descending    2

/*
*   PATH Class Type
*/
#define Path_Type_Local      0
#define Path_Type_nonLocal   1


/*
*   PATH Trace Status
*/
#define Trace_NoCurrent      0
#define Trace_Initiated      1
#define Trace_Propagated     2
#define Trace_Terminated     3

/*
*   Path Status
*/
#define Path_Status_Wrapped  0
#define Path_Status_Thru     1


/*****************************************************************************
    PORT Definitions
*****************************************************************************/

/*
*   PC_Type & PC_Neighbor Values
*/
#define PC_Type_A        0
#define PC_Type_B        1
#define PC_Type_S        2
#define PC_Type_M        3
#define PC_Type_Unknown  4
#define PC_Type_None     PC_Type_Unknown

/*
*   Connection State Values
*/
#define Connect_Disabled     0   /* PC0:OFF */
#define Connect_Connecting   1   /* !PC8:ACTIVE & !PC0:OFF !PC2:TRACE
                                 & PC_Withhold = None */
#define Connect_Standby      2   /* PC_Withhold != None */
#define Connect_Active       3   /* PC8:ACTIVE */

/*
*   PORT Connection Policies.
*/
#define PC_MAC_None          0
#define PC_MAC_LCT           0x01
#define PC_MAC_Loop          0x02
#define PC_MAC_Placement     0x04

/*
*   PC_Withhold Values
*/
#define PC_WH_None       0   /* no withholding */
#define PC_WH_M_to_M     1   /* PHY M to PHY M */
#define PC_WH_Other      2   /* Other incompatible PHY types */
```

```
/*
*   Maintenance Line States
*/
#define Maint_QLS    0
#define Maint_ILS    1
#define Maint_MLS    2
#define Maint_HLS    3
#define Maint_PDR    4


/*
*   Limits on LER ranges
*/
#define LER_MIN 4
#define LER_MAX 15

/*
*   PORT FOTX Class
*/
#define FOTX_Multimode       0
#define FOTX_Single_Mode1    1
#define FOTX_Single_Mode2    2
#define FOTX_SONET           3

/*
*   PORT Actions
*/
#define PORTAction_Maint     0
#define PORTAction_Enable    1
#define PORTAction_Disable   2
#define PORTAction_Start     3
#define PORTAction_Stop      4


/************************************************************************
    ATTACHMENT Definitions
************************************************************************/

/*
*   ATTACHMENT Classes
*/
#define Attachment_Class_Single          0
#define Attachment_Class_Dual            1
#define Attachment_Class_Concentrator    2


/************************************************************************
    CMT Definitions
************************************************************************/

/*
*   Path_Test Values
*/
#define PT_None      0
#define PT_Testing   1
#define PT_Passed    2
#define PT_Failed    3
#define PT_Pending   4
#define PT_Exiting   5

/*
*   PC_Mode Values
*/
#define PC_Mode_Peer     0
#define PC_Mode_Tree     1
#define PC_Mode_None     2
```

```
/*
*   Line States
*   These lines define the constant names used in the CSP code.
*   The actual values for the line states are defined in smtdefs.h.
*/
#define ALS         (Active_Line_State)
#define HLS         (Halt_Line_State)
#define ILS         (Idle_Line_State)
#define MLS         (Master_Line_State)
#define NLS         (Noise_Line_State)
#define PDRLS       (Transmit_PHY_Data_Request)
#define QLS         (Quiet_Line_State)
#define SILS        (Super_Idle_Line_State)

/*
*   RMT Duplicate Address Policies
*/
#define RMT_DUP_CHANGE   1       /* perform change actions */
#define RMT_DUP_JAM_A    2       /* perform jam actions */
#define RMT_DUP_JAM_B    3       /* perform jam actions */
#define RMT_DUP_LEAVE    4       /* leave ring */


/*************************************************************************
    FBM Definitions
*************************************************************************/

/*
*   Known Address Formats
*/
    /* FDDI broadcast address X'FF FF FF FF FF FF' */
#define BROADCAST_ADDRESS    "\377\377\377\377\377\377"

    /* FDDI Null address X'00 00 00 00 00 00' */
#define NULL_ADDRESS         "\0\0\0\0\0\0"
#define UNKNOWN_ADDRESS      NULL_ADDRESS

    /* Directed beacon for RMT X'01 80 C2 00 01 00' (canonical) */
    /*                        X'80 01 43 00 80 00' (FDDI) */
#define DIR_BCN_ADDRESS      "\200\001\103\000\200\000"

    /* SRF Multicast address X'01 80 C2 00 01 10' (canonical) */
    /*                       X'80 01 43 00 80 08' (FDDI) */
#define SRF_MULTICAST_ADDRESS    "\200\001\103\000\200\010"

/*
*   Reason Denied Codes
*/
#define RC_FRAME_CLASS          ((uInt32) 0x01)
#define RC_FRAME_VERSION        ((uInt32) 0x02)
#define RC_SUCCESS              ((uInt32) 0x03)
#define RC_BAD_SET_COUNT        ((uInt32) 0x04)
#define RC_READ_ONLY            ((uInt32) 0x05)
#define RC_NO_PARAM             ((uInt32) 0x06)
#define RC_NO_MORE              ((uInt32) 0x07)
#define RC_OUT_OF_RANGE         ((uInt32) 0x08)
#define RC_NOT_AUTHORIZED       ((uInt32) 0x09)
#define RC_LENGTH_ERROR         ((uInt32) 0x0A)
#define RC_FRAME_TOO_LONG       ((uInt32) 0x0B)
#define RC_ILLEGAL_PARAMETER    ((uInt32) 0x0C)

/* XLNT Manager value */
#define RC_ERROR_UNKNOWN        ((uInt32) 0xFE)
#define RC_BUFFER_ERROR         ((uInt32) 0xFF)
```

```
/*
*   Node Class of Station Descriptor Parameter
*   The type of station we are can be one of two things, either a
*   station or a concentrator.  The values for these are defined
*   below.
*/
#define SMT_Type_Station 0
#define SMT_Type_Concentrator    1

/*
*   Topology Bit Assignments of Station State Parameter
*   The station's current topology is indicated by which bit is set.
*/
#define Topology_Thru                0x00
#define Topology_Wrapped             0x01
#define Topology_Unrooted            0x02
#define Topology_Twisted_AA          0x04
#define Topology_Twisted_BB          0x08
#define Topology_Rooted_Station      0x10
#define Topology_SRF                 0x20

/*
*   Duplicate Address Bit Assignments of Station State Parameter
*   MAC's current duplicate address status for the
*   Station State parameter.
*/
#define DuplAddr_My_Duplicate        0x01
#define DuplAddr_My_UNA_Duplicate    0x02

/*
*   Maximum Frame Size (in bytes)
*/
#define MAX_FRAME_SIZE      4500

/*
*   Maximum SMT INFO field size (in bytes)
*/
#define MAX_SMT_INFO_LEN    4458

/*
*   Maximum Interval Between SRFs (in seconds).
*/
#define MAX_SRF_INTERVAL    32

/*
*   Frame Status Values
*   Defined in smtdefs.h.
*/
#define E_Indicator  E_Bit_Position
#define A_Indicator  A_Bit_Position
#define C_Indicator  C_Bit_Position


/*****************************************************************************
*   XLNT Manager Standard Defined Values
*****************************************************************************/

/*
*   Attach PORT Indexes
*/
#define PHY_A   0
#define PHY_B   1
#define PHY_S   0
```

```
/*
*   Station MAC Indexes
*/
#define PRIMARY_MAC      0
#define SECONDARY_MAC    1
#define LOCAL_MAC        2
#define MAC_P            PRIMARY_MAC
#define MAC_S            SECONDARY_MAC
#define MAC_L            LOCAL_MAC

/* These values are left for compatibility with older software */
#define MAC_1    0
#define MAC_2    1

/*
*   PATH Indexes
*/
#define PATH_P  0
#define PATH_S  1
#define PATH_L  2

/*
*   MIB Attach PORT Indexes
*/
#define MIB_PHY_A    1
#define MIB_PHY_B    2
#define MIB_PHY_S    1

/*
*   MIB MAC Indexes
*/
#define MIB_MAC_P    1
#define MIB_MAC_S    2
#define MIB_MAC_L    3

/*
*   Entity Types
*
*   These are arbitrary assignments until the SMT committee sets
*   the proper values.
*/
#define PHY_TYPE     0
#define PORT_TYPE    PHY_TYPE
#define MAC_TYPE     1
```

```
/**********************************************************************
    Management Information Base Header File

    MIB structure definition.

    File:       mibtypes.h

    This file contains type definitions used by the Management
    Information Base (MIB).  The MIB is accessed through
    get, set, add and change routines.  It contains or has access
    to all SMT attributes within a particular station.

    Modification History:

    *** CREATED FOR SMT 6.2 ***

**********************************************************************/

/**********************************************************************
*
*   Typedefs for the attribute groups. Groups are based on the
*   format as defined in the SMT standard. The member names are
*   the attribute names with the object portion deleted (e.g.,
*   fddiSMTStationId becomes SMTStationId). The group names use the
*   attribute group names with the "fddi" portion deleted.
*
**********************************************************************/

/**********************************************************************
*   SMT Group Types
**********************************************************************/

/*
*   fddiSMTStationIdGrp  {fddiSMT 10}
*/
typedef struct SMTStationIdGrpStruct {
    SMTStationIdType        StationId;          /* the SMT station id */
    uInt16                  OpVersionId;        /* version of SMT */
    uInt16                  HiVersionId;        /* highest version supported */
    uInt16                  LoVersionId;        /* lowest version supproted */
    SMTManufacturerDataType ManufacturerData;   /* manufacturer's data */
    SMTUserDataType         UserData;           /* user data */
} SMTStationIdGrpType;

/*
*   fddiSMTStationConfigGrp  {fddiSMT 20}
*/
typedef struct SMTStationConfigGrpStruct {
    uChar       MAC_Ct;             /* number of MACs */
    uChar       NonMaster_Ct;       /* number of attachments */
    uChar       Master_Ct;          /* number of masters */
    uChar       PathsAvailable;     /* available paths in stn */
    uInt16      ConfigCapabilities; /* configuration options */
    uInt16      ConfigPolicy;       /* configuration selected */
    uInt16      ConnectionPolicy;   /* connections allowed */
    uInt32      ReportLimit;        /* SRF limit */
    uInt16      T_Notify;           /* NIF NN interval */
    uChar       StatusReporting;    /* SRF flag */
} SMTStationConfigGrpType;
```

```
/*
*   fddiSMTStatusGrp     {fddiSMT 40}
*/
typedef struct SMTStatusGrpStruct {
    uChar       ECMState;               /* current ECM state */
    uChar       CF_State;               /* current config state */
    uInt16      HoldState;              /* current Hold condition */
    uChar       RemoteDisconnectFlag;   /* remotely disconnected */
} SMTStatusGrpType;

/*
*   fddiSMTMIBOperationGrp   {fddiSMT 50}
*/
typedef struct SMTMIBOperationGrpStruct {
    SMTTimeStamp     MsgTimeStamp;          /* station time stamp */
    SMTTimeStamp     TransitionTimeStamp;   /* SRF time stamp */
    SetCountType     SetCount;              /* PMF support */
    SMTStationIdType
            LastSetStationId;               /* last station to set MIB */
} SMTMIBOperationGrpType;

/*****************************************************************************
*   MAC Group Types
*****************************************************************************/

/*
*   fddiMACCapabilitiesGrp   {fddiMAC 10}
*/
typedef struct MACCapabilitiesGrpStruct {
    uInt16      FrameStatusCapabilities;    /* AC indicator capabilities */
    uInt16      BridgeFunction;             /* bridging type */
    uInt32      T_MaxGreatestLowerBound;    /* lowest T_Max */
    uInt32      TVXGreatestLowerBound;      /* lowext TVX */
} MACCapabilitiesGrpType;

/*
*   fddiMACConfigGrp     {fddiMAC 20}
*/
typedef struct MACConfigGrpStruct {
    uChar       PathsAvailable;         /* usable MAC paths */
    uInt16      CurrentPath;            /* path for MAC */
    MACAddr48   UpstreamNbr;            /* MAC's upstream neighbor */
    MACAddr48   DownstreamNbr;          /* MAC's dwnstrm neighbor */
    MACAddr48   OldUpstreamNbr;         /* MAC's previous nbr */
    MACAddr48   OldDownstreamNbr;       /* MAC's pervious nbr */
    uChar       RootConcentratorMAC;    /* flag if root */
    uInt16      Dup_Addr_Test;          /* dup addr test status */
    uChar       PathsRequested;         /* preferred path for MAC */
    uInt16      DownstreamPORTType;     /* PC-Type of dwnstrm prt */
} MACConfigGrpType;

/*
*   fddiMACAddressGrp    {fddiMAC 40}
*/
typedef struct MACAddressGrpStruct {
    MACAddr48   SMTAddress;         /* MAC's 48-bit address */
    uChar       *LongAliases;       /* list of long aliases */
    uChar       *ShortAliases;      /* list of short aliases */
    uChar       *LongGrpAddrs;      /* list of long group addrs */
    uChar       *ShortGrpAddrs;     /* list of short group addrs */
} MACAddressGrpType;
```

```
/*
*   fddiMACOperationGrp  {fddiMAC 50}
*/
typedef struct MACOperationGrpStruct {
    uInt32      T_Req;
    uInt32      T_Neg;
    uInt32      T_Max;
    uInt32      TvxValue;
    uInt32      T_Min;
    uInt32      T_Pri0;
    uInt32      T_Pri1;
    uInt32      T_Pri2;
    uInt32      T_Pri3;
    uInt32      T_Pri4;
    uInt32      T_Pri5;
    uInt32      T_Pri6;
    uInt16      FrameStatus;        /* MAC's current status */
} MACOperationGrpType;

/*
*   fddiMACCountersGrp   {fddiMAC 70}
*/
typedef struct MACCountersGrpStruct {
    uInt32      Frame_Ct;
    uInt32      Copied_Ct;
    uInt32      Transmit_Ct;
    uInt32      Token_Ct;
} MACCountersGrpType;

/*
*   fddiMACErrorCtrsGrp  {fddiMAC 80}
*/
typedef struct MACErrorCtrsGrpStruct {
    uInt32      Error_Ct;
    uInt32      Lost_Ct;
    uInt32      TvxExpired_Ct;
    uInt32      NotCopied_Ct;
    uInt16      Late_Ct;
    uInt32      RingOp_Ct;
} MACErrorCtrsGrpType;

/*
*   fddiMACFrameErrorConditionGrp    {fddiMAC 90}
*/
typedef struct MACFrameErrorConditionGrpStruct {
    uInt32      BaseFrame_Ct;
    uInt32      BaseError_Ct;
    uInt32      BaseLost_Ct;
    SMTTimeStamp    BaseTimeFrameError;
    uInt16      FrameErrorThreshold;
    uInt16      FrameErrorRatio;
} MACFrameErrorConditionGrpType;

/*
*   fddiMACNotCopiedConditionGrp {fddiMAC 100}
*/
typedef struct MACNotCopiedConditionGrpStruct {
    uInt32          BaseNotCopied_Ct;
    SMTTimeStamp    BaseTimeNotCopied;
    uInt16          NotCopiedThreshold;
    uInt32          BaseCopied_Ct;
    uInt16          NotCopiedRatio;
} MACNotCopiedConditionGrpType;
```

```
/*
*   fddiMACStatusGrp     {fddiMAC 110}
*/
typedef struct MACStatusGrpStruct {
    uChar       RMTState;               /* current RMT state */
    uChar       Da_Flag;                /* dup addr flag */
    uChar       UnaDa_Flag;             /* dup addr from UNA */
    uChar       FrameErrorCondition;    /* current condition */
    uChar       NotCopiedCondition;     /* current condition */
    uChar       LLCServiceAvailable;    /* LLC service enabled */
} MACStatusGrpType;

/*
*   fddiMACRootMACStatusGrp  {fddiMAC 120}
*/
typedef struct MACRootMACStatusGrpStruct {
    uChar       MasterSlaveLoopStatus;  /* loop detection stat */
    uChar       RootMACDownStreamPORTType;
    uInt16      RootMACCurrentPath;  /* path for root MAC */
} MACRootMACStatusGrpType;

/*****************************************************************************
*   PATH Class Group Types
*****************************************************************************/

/*
*   fddiPATHClassConfigGrp   {fddiPATHClass 10}
*/
typedef struct ClassConfigGrpStruct {
    uInt32      Trace_MaxExpiration;    /* value of Trace_Max */
    uInt32      TVXLowerBound;          /* lower bound of TVX */
    uInt32      T_MaxLowerBound;        /* lower bound of T_Max */
} ClassConfigGrpType;

/*****************************************************************************
*   PATH Class PATH Group Types
*****************************************************************************/

/*
*   fddiPATHClassPATHConfigGrp   {fddiPATHClassPATH 10}
*/
typedef struct ClassPATHConfigGrpStruct {
    uInt16      Type;                   /* which path */
    uInt16      PORTOrder;              /* ascending/descending */
    uInt32      RingLatency;            /* latency on this path */
    uInt16      TraceStatus;            /* current trace status */
    uInt32      Sba;                    /* SBA for path */
    uInt16      SbaOverhead;            /* overhead for path */
    uInt16      Status;                 /* wrapped/through */
    PathConfigType  Configuration;      /* path configuration */
    uInt32      T_Rmode;                /* restricted dialog limit */
} ClassPATHConfigGrpType;
```

```
/*************************************************************************
*   PORT Group Types
*************************************************************************/

/*
*   fddiPORTConfigGrp    {fddiPORT 10}
*/
typedef struct PORTConfigGrpStruct {
    uChar       PC_Type;                /* type of PORT */
    uChar       PC_Neighbor;            /* type of neighbor PORT */
    uChar       ConnectionPolicies;     /* requested policies */
    uChar       RemoteMACIndicated;     /* 1 if True, 0 False */
    uChar       CE_State;               /* config element state */
    uChar       PathsRequested;         /* requested path for S/M */
    uInt16      MACPlacement;           /* MAC index of upstream MAC */
    uChar       AvailablePaths;         /* paths usable for S/M */
    uInt32      MACLoop_Time;           /* value of T_Next(9) */
    uChar       FotxClass;              /* optic xmitter class */
} PORTConfigGrpType;

/*
*   fddiPORTOperationGrp     {fddiPORT 30}
*/
typedef struct PORTOperationGrpStruct {
    uChar       MaintLineState;         /* current maint LS */
    uInt32      TB_Max;                 /* value of TB_Max */
    uChar       BS_Flag;                /* 1 True, 0 False */
} PORTOperationGrpType;

/*
*   fddiPORTErrorCtrsGrp     {fddiPORT 40}
*/
typedef struct PORTErrorCtrsGrpStruct {
    uInt32      EBErr_Ct;       /* EB overflow count */
    uInt32      LCTFail_Ct;     /* # fails for LCT */
} PORTErrorCtrsGrpType;

/*
*   fddiPORTLerGrp   {fddiPORT 50}
*/
typedef struct PORTLerGrpStruct {
    uChar       Ler_Estimate;       /* exponent of estimate */
    uInt32      Lem_Reject_Ct;      /* # times link rejected */
    uInt32      Lem_Ct;             /* # errors detected */
    uChar       BaseLer_Estimate;
    uInt32      BaseLem_Reject_Ct;
    uInt32      BaseLem_Ct;
    SMTTimeStamp    BaseLer_TimeStamp;
    uChar       Ler_Cutoff;     /* level for cutoff */
    uChar       Ler_Alarm;      /* level for alarm */
} PORTLerGrpType;

/*
*   fddiPORTStatusGrp    {fddiPORT 50}
*/
typedef struct PORTStatusGrpStruct {
    uInt16      ConnectState;       /* current connect state */
    uChar       PCMState;           /* state of PCM */
    uChar       PC_Withhold;        /* withhold reason */
    uChar       LerCondition;       /* current condition state */
} PORTStatusGrpType;
```

```
/*****************************************************************************
*   ATTACHMENT Group Types
*****************************************************************************/

/*
*   fddiATTACHMENTConfigGrp  {fddiATTACHMENT 10}
*/
typedef struct ATTACHMENTConfigGrpStruct {
    uInt16      Class;                  /* dual, single, conc. */
    uChar       OpticalBypassPresent;   /* 1 True, 0 False */
    uInt32      I_MaxExpiration;        /* I_Max value */
    uChar       InsertedStatus;         /* 1 True, 0 False */
    uChar       InsertPolicy;           /* 1 insert, 0 don't insert */
} ATTACHMENTConfigGrpType;

/*****************************************************************************
*   XDI Group Types
*****************************************************************************/

typedef struct XDISMTGrpStruct {
    uChar       BothWrapCapability;
    uChar       BothWrapPolicy;
    uChar       Topology;
    uInt16      OutIndex1;
    uInt16      OutIndex2;
    uInt16      Port_Ct;
    uChar       SB_Flag;
} XDISMTGrpType;

typedef struct XDIMACGrpStruct {
    uChar       RMTDupPolicy1;
    uChar       RMTDupPolicy2;
    uChar       NeighborChange;
    MACAddr48   DataAddress;
    uChar       Operational;
} XDIMACGrpType;

typedef struct XDIPORTGrpStruct {
    uChar       Operational;
    uInt32      BaseEBError_Ct;
} XDIPORTGrpType;

#endif  /* _IF_PRG_H */
```

This page is intentionallly left blank.

APPENDIX C
BASE ADDRESS JUMPER SETTINGS

As discussed in the installation section (p. 2-5), pins 1 - 7 of jumper field JA11 are used to set the base address of the 4211's 512-byte short I/O space. The following table shows the jumper settings for all possible base addresses. To locate the JA11 jumper field, refer to the board drawing on p. 2-2.

Table C-1 . Base Address Jumper Settings
(Jumper Field JA11, Pins 1 - 7)

Address	Pin 7	Pin 6	Pin 5	Pin 4	Pin 3	Pin 2	Pin 1
0x0000	IN	IN	IN	IN	IN	IN	IN
0x0200	IN	IN	IN	IN	IN	IN	OUT
0x0400	IN	IN	IN	IN	IN	OUT	IN
0x0600	IN	IN	IN	IN	IN	OUT	OUT
0x0800	IN	IN	IN	IN	OUT	IN	IN
0x0A00	IN	IN	IN	IN	OUT	IN	OUT
0x0C00	IN	IN	IN	IN	OUT	OUT	IN
0x0E00	IN	IN	IN	IN	OUT	OUT	OUT
0x1000	IN	IN	IN	OUT	IN	IN	IN
0x1200	IN	IN	IN	OUT	IN	IN	OUT
0x1400	IN	IN	IN	OUT	IN	OUT	IN
0x1600	IN	IN	IN	OUT	IN	OUT	OUT
0x1800	IN	IN	IN	OUT	OUT	IN	IN
0x1A00	IN	IN	IN	OUT	OUT	IN	OUT
0x1C00	IN	IN	IN	OUT	OUT	OUT	IN
0x1E00	IN	IN	IN	OUT	OUT	OUT	OUT
0x2000	IN	IN	OUT	IN	IN	IN	IN
0x2200	IN	IN	OUT	IN	IN	IN	OUT
0x2400	IN	IN	OUT	IN	IN	OUT	IN
0x2600	IN	IN	OUT	IN	IN	OUT	OUT
0x2800	IN	IN	OUT	IN	OUT	IN	IN
0x2A00	IN	IN	OUT	IN	OUT	IN	OUT
0x2C00	IN	IN	OUT	IN	OUT	OUT	IN
0x2E00	IN	IN	OUT	IN	OUT	OUT	OUT
0x3000	IN	IN	OUT	OUT	IN	IN	IN
0x3200	IN	IN	OUT	OUT	IN	IN	OUT
0x3400	IN	IN	OUT	OUT	IN	OUT	IN
0x3600	IN	IN	OUT	OUT	IN	OUT	OUT
0x3800	IN	IN	OUT	OUT	OUT	IN	IN

(continued on next page)

(Table C–1, cont.)

Address	Pin 7	Pin 6	Pin 5	Pin 4	Pin 3	Pin 2	Pin 1
0x3A00	IN	IN	OUT	OUT	OUT	IN	OUT
0x3C00	IN	IN	OUT	OUT	OUT	OUT	IN
0x3E00	IN	IN	OUT	OUT	OUT	OUT	OUT
0x4000	IN	OUT	IN	IN	IN	IN	IN
0x4200	IN	OUT	IN	IN	IN	IN	OUT
0x4400	IN	OUT	IN	IN	IN	OUT	IN
0x4600	IN	OUT	IN	IN	IN	OUT	OUT
0x4800	IN	OUT	IN	IN	OUT	IN	IN
0x4A00	IN	OUT	IN	IN	OUT	IN	OUT
0x4C00	IN	OUT	IN	IN	OUT	OUT	IN
0x4E00	IN	OUT	IN	IN	OUT	OUT	OUT
0x5000	IN	OUT	IN	OUT	IN	IN	IN
0x5200	IN	OUT	IN	OUT	IN	IN	OUT
0x5400	IN	OUT	IN	OUT	IN	OUT	IN
0x5600	IN	OUT	IN	OUT	IN	OUT	OUT
0x5800	IN	OUT	IN	OUT	OUT	IN	IN
0x5A00	IN	OUT	IN	OUT	OUT	IN	OUT
0x5C00	IN	OUT	IN	OUT	OUT	OUT	IN
0x5E00	IN	OUT	IN	OUT	OUT	OUT	OUT
0x6000	IN	OUT	OUT	IN	IN	IN	IN
0x6200	IN	OUT	OUT	IN	IN	IN	OUT
0x6400	IN	OUT	OUT	IN	IN	OUT	IN
0x6600	IN	OUT	OUT	IN	IN	OUT	OUT
0x6800	IN	OUT	OUT	IN	OUT	IN	IN
0x6A00	IN	OUT	OUT	IN	OUT	IN	OUT
0x6C00	IN	OUT	OUT	IN	OUT	OUT	IN
0x6E00	IN	OUT	OUT	IN	OUT	OUT	OUT
0x7000	IN	OUT	OUT	OUT	IN	IN	IN
0x7200	IN	OUT	OUT	OUT	IN	IN	OUT
0x7400	IN	OUT	OUT	OUT	IN	OUT	IN
0x7600	IN	OUT	OUT	OUT	IN	OUT	OUT
0x7800	IN	OUT	OUT	OUT	OUT	IN	IN
0x7A00	IN	OUT	OUT	OUT	OUT	IN	OUT
0x7C00	IN	OUT	OUT	OUT	OUT	OUT	IN
0x7E00	IN	OUT	OUT	OUT	OUT	OUT	OUT
0x8000	OUT	IN	IN	IN	IN	IN	IN
0x8200	OUT	IN	IN	IN	IN	IN	OUT
0x8400	OUT	IN	IN	IN	IN	OUT	IN
0x8600	OUT	IN	IN	IN	IN	OUT	OUT
0x8800	OUT	IN	IN	IN	OUT	IN	IN
0x8A00	OUT	IN	IN	IN	OUT	IN	OUT
0x8C00	OUT	IN	IN	IN	OUT	OUT	IN
0x8E00	OUT	IN	IN	IN	OUT	OUT	OUT

(continued on next page)

(Table C–1, cont.)

Address	Pin 7	Pin 6	Pin 5	Pin 4	Pin 3	Pin 2	Pin 1
0x9000	OUT	IN	IN	OUT	IN	IN	IN
0x9200	OUT	IN	IN	OUT	IN	IN	OUT
0x9400	OUT	IN	IN	OUT	IN	OUT	IN
0x9600	OUT	IN	IN	OUT	IN	OUT	OUT
0x9800	OUT	IN	IN	OUT	OUT	IN	IN
0x9A00	OUT	IN	IN	OUT	OUT	IN	OUT
0x9C00	OUT	IN	IN	OUT	OUT	OUT	IN
0x9E00	OUT	IN	IN	OUT	OUT	OUT	OUT
0xA000	OUT	IN	OUT	IN	IN	IN	IN
0xA200	OUT	IN	OUT	IN	IN	IN	OUT
0xA400	OUT	IN	OUT	IN	IN	OUT	IN
0xA600	OUT	IN	OUT	IN	IN	OUT	OUT
0xA800	OUT	IN	OUT	IN	OUT	IN	IN
0xAA00	OUT	IN	OUT	IN	OUT	IN	OUT
0xAC00	OUT	IN	OUT	IN	OUT	OUT	IN
0xAE00	OUT	IN	OUT	IN	OUT	OUT	OUT
0xB000	OUT	IN	OUT	OUT	IN	IN	IN
0xB200	OUT	IN	OUT	OUT	IN	IN	OUT
0xB400	OUT	IN	OUT	OUT	IN	OUT	IN
0xB600	OUT	IN	OUT	OUT	IN	OUT	OUT
0xB800	OUT	IN	OUT	OUT	OUT	IN	IN
0xBA00	OUT	IN	OUT	OUT	OUT	IN	OUT
0xBC00	OUT	IN	OUT	OUT	OUT	OUT	IN
0xBE00	OUT	IN	OUT	OUT	OUT	OUT	OUT
0xC000	OUT	OUT	IN	IN	IN	IN	IN
0xC200	OUT	OUT	IN	IN	IN	IN	OUT
0xC400	OUT	OUT	IN	IN	IN	OUT	IN
0xC600	OUT	OUT	IN	IN	IN	OUT	OUT
0xC800	OUT	OUT	IN	IN	OUT	IN	IN
0xCA00	OUT	OUT	IN	IN	OUT	IN	OUT
0xCC00	OUT	OUT	IN	IN	OUT	OUT	IN
0xCE00	OUT	OUT	IN	IN	OUT	OUT	OUT
0xD000	OUT	OUT	IN	OUT	IN	IN	IN
0xD200	OUT	OUT	IN	OUT	IN	IN	OUT
0xD400	OUT	OUT	IN	OUT	IN	OUT	IN
0xD600	OUT	OUT	IN	OUT	IN	OUT	OUT
0xD800	OUT	OUT	IN	OUT	OUT	IN	IN
0xDA00	OUT	OUT	IN	OUT	OUT	IN	OUT
0xDC00	OUT	OUT	IN	OUT	OUT	OUT	IN
0xDE00	OUT	OUT	IN	OUT	OUT	OUT	OUT
0xE000	OUT	OUT	OUT	IN	IN	IN	IN
0xE200	OUT	OUT	OUT	IN	IN	IN	OUT
0xE400	OUT	OUT	OUT	IN	IN	OUT	IN

(continued on next page)

(Table C–1, cont.)

Address	Pin 7	Pin 6	Pin 5	Pin 4	Pin 3	Pin 2	Pin 1
0xE600	OUT	OUT	OUT	IN	IN	OUT	OUT
0xE800	OUT	OUT	OUT	IN	OUT	IN	IN
0xEA00	OUT	OUT	OUT	IN	OUT	IN	OUT
0xEC00	OUT	OUT	OUT	IN	OUT	OUT	IN
0xEE00	OUT	OUT	OUT	IN	OUT	OUT	OUT
0xF000	OUT	OUT	OUT	OUT	IN	IN	IN
0xF200	OUT	OUT	OUT	OUT	IN	IN	OUT
0xF400	OUT	OUT	OUT	OUT	IN	OUT	IN
0xF600	OUT	OUT	OUT	OUT	IN	OUT	OUT
0xF800	OUT	OUT	OUT	OUT	OUT	IN	IN
0xFA00	OUT	OUT	OUT	OUT	OUT	IN	OUT
0xFC00	OUT	OUT	OUT	OUT	OUT	OUT	IN
0xFE00	OUT	OUT	OUT	OUT	OUT	OUT	OUT

APPENDIX D
DOWNLOADING CODE

This appendix describes how to download and execute code on the V/FDDI 4211 Peregrine. Most 4211 applications *do not* require users to be familiar with this feature in order to integrate the controller into the host system.

The host system can download code to 4211 firmware via the board's Common Boot interface. Applications of the download feature include:

- making firmware upgrades in the field
- booting an interface other than the native RC Interface as documented in Chapter 3 of the *Common Boot & RC Interface User's Guide* (UG99999-000)

The sequence of events for downloading and executing code is as follows:

1. The host brings up the Common Boot interface on the controller. As discussed in Chapter 2 of the *Common Boot & RC Interface User's Guide*, the controller starts up in Common Boot mode each time it is powered up. The Command Response Word should contain the ASCII value "CBOK".

2. The host executes a series of POKE commands to download code into controller memory. Figure D-1 shows how the fields of this Common Boot command are filled in.

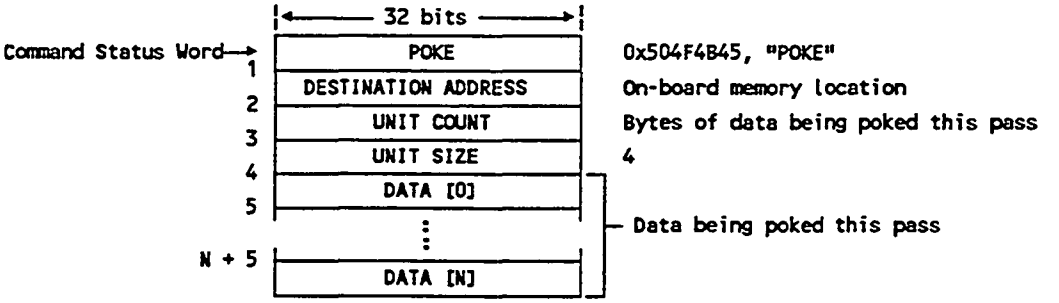


Figure D–1 . Using POKE Command to Download Code

The meaning of the fields in Figure D-1 is as follows.

- **Command Status Word**

  This field contains the ASCII value POKE (0x504F4B45). This field is filled in *after* the other command fields (described below) have been filled in.

- **Destination Address**

  On the first pass, this field contains the controller-specific value CB_BOARD_LOCATION. This value is 0x80002000 for the V/FDDI 4211 Peregrine. As successive batches of data are poked onboard, this address is incremented to point past previously downloaded data.

- **Unit Count**

  This is the number of bytes being poked this pass. The maximum byte count is 208 bytes. (This is due to ioctl-related transfer restrictions in the Unix 4.3 BSD operating system implementation on Sun machines).

- **Unit Size**

  This field sets the width (in bytes) of onboard memory-to-memory transfers. It contains the value 4 to specify longword transfers.

- **Data**

  This is the data being poked into memory this pass. As noted previously, no more than 208 bytes of data can be poked at once. Moreover, the V/FDDI 4211 Peregrine expects the data to be word- or longword-aligned (no byte transfers). Under *no* circumstances should data be written outside the 4211's 512-byte short I/O space.

3. The host executes a BOOT command with the following format:

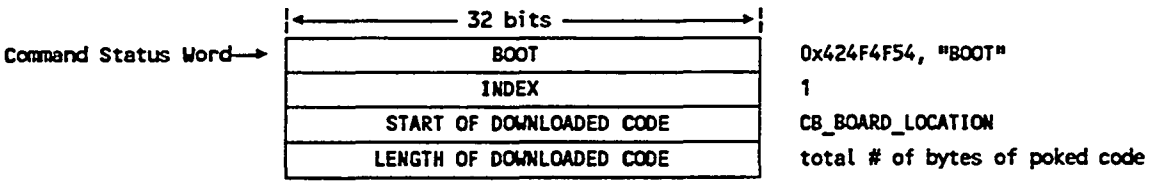


Figure D–2 . Using BOOT to Run Downloaded Code

4. The controller moves the downloaded code to static RAM, transfers execution to the start of SRAM, and performs its bootstrap code. It then returns to Common Boot mode. The Command Status Word will contain CBOK. The downloaded code remains in SRAM until the board is reset.

The host may then issue a BOOT command with an index of 0 to boot the RC Interface. Do *not* reset the board before booting. Doing so causes the downloaded code to be replaced by the default code in controller firmware.

INDEX

OPEN SYSTEMS CONTROLLERS™

Disk • Tape • Networking

13800 Senlac • Dallas, Texas 75234 • (214) 919-9000 • FAX: (214) 919-9200 • NASDAQ-NMS:INPH

Interphase International

Astral House, Granville Way • Bicester, Oxon OX6 OJT • (01144) 869 321222 • FAX (01144) 869-247720