PCI/EISA Bus-Master Adapter Driver Technical Reference

Members of the 3Com EtherLink® III and Fast EtherLink families of adapters

For 3Com User Group Information
1-800-NET-3Com
or your local 3Com office

Manual Part Number 09-0681-001B
Printed May 1995. Printed in the U.S.A.

3Com Corporation
5400 Bayfront Plaza
Santa Clara,
California, USA
95052-8145

3Com Corporation reserves the right to revise this documentation and to make changes in content from time to time without obligation on the part of 3Com Corporation to provide notification of such revision or change.

3Com Corporation provides this documentation without warranty of any kind, either implied or expressed, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. 3Com may make improvements or changes in the product(s) and/or the program(s) described in this documentation at any time.

UNITED STATES GOVERNMENT LEGENDS:
If you are a United States government agency, then this documentation and the software described herein are provided to you subject to the following restricted rights:

For units of the Department of Defense:
Restricted Rights Legend: Use, duplication or disclosure by the Government is subject to restrictions as set forth in subparagraph (c) (1) (ii) for restricted Rights in Technical Data and Computer Software clause at 48 C.F.R. 52.227-7013. 3Com Corporation, 5400 Bayfront Plaza, Santa Clara, California 95052-8145.

For civilian agencies:
Restricted Rights Legend: Use, reproduction or disclosure is subject to restrictions set forth in subparagraph (a) through (d) of the Commercial Computer Software - Restricted Rights Clause at 48 C.F.R. 52.227-19 and the limitations set forth in 3Com Corporation's standard commercial agreement for the software. Unpublished rights reserved under the copyright laws of the United States.

Unless otherwise indicated, 3Com registered trademarks are registered in the United States and may or may not be registered in other countries.

3Com and EtherLink are registered trademarks of 3Com Corporation.

Contents

Chapter 4 Register Listings

Chapter 5 Adapter Configuration

Figures

Tables

Chapter 1
Introduction

This manual defines the programming interface supported by first-generation bus mastering adapters for the PCI and EISA buses running at speeds of 10 or 100 Mbps.

This manual is for software engineers and test engineers to use as a reference in writing device drivers, diagnostics, and production test software. PCI and EISA bus master adapters share many of the same characteristics. In this manual, differences are clearly indicated as “PCI Only” or “EISA Only.” If there is no such indication, the information is valid for either type of adapter.

Summary of Features

PCI/EISA bus master adapters are based upon the EtherLink® III adapter architecture. PCI/EISA bus master adapters have a programming interface similar to that of the first-generation programmed input/output (PIO) and later adapters, augmented by a single-fragment bus master data transfer mode.

The PCI/EISA bus-master bus architecture adds specifications for several optional extensions to the PIO architecture. Bus master adapters indicate their support for these extensions to the software via a new Capabilities Word in the EEPROM.

The PCI/EISA bus master adapter architecture includes support for the 100BASE-TX/FX 100 Mbps signaling standard and the three 10 Mbps Ethernet signaling standards: 10BASE-T, 10BASE2, and 10BASE5. The PCI/EISA bus master architecture can also support other 10 Mbps and 100 Mbps signaling schemes (such as 100BASE-T4) using the Media Independent Interface. Adapters based upon the PCI/EISA bus master architecture will include various combinations of these media ports. Drivers may be written to support all combinations of these available media ports automatically.

New features supported by all PCI/EISA bus master adapters are CRC passthrough, the `TxDone` command, large packet handling, an extended deference mechanism, and individual address bit-masking.

PCI/EISA bus master adapters support a variable amount of packet buffer RAM. Since different adapters have differing amounts of RAM installed, drivers written for these adapters must allow for different amounts. The allocation of RAM between the receive and transmit functions is configurable through the **InternalConfig** register.

PCI/EISA Bus-Master Adapter Features

PCI/EISA bus master architecture uses EtherLink III PIO architecture with fragment bus master extensions. To support bus master operations, a number of registers have been added in Window 7.

This architecture supports the following:

- 100 Mbps signaling standard (100BASE-TX)
- 10 Mbps signaling standards (10BASE-T, 10BASE2, and 10BASE5)
- 100BASE-T4 signaling standard through the Media Independent Interface (MII)
- Up to 128 KB of packet buffer RAM, which can be divided 1:1, 3:1, or 5:3 between the receive and transmit functions
- Packet sizes up to 4490 bytes
- CRC passthrough for bridging applications, via a static configuration option on receive and a frame-by-frame basis on transmit
- The `TxDone` command, to eliminate the need to dword pad the transmit data
- 8 K, 16 K, 32 K or 64 K BIOS ROM

Nomenclature

The following nomenclature is used throughout this manual:

Indications	The reporting of any interesting event on the adapter. Any indication may be configured to cause an interrupt.
Interrupts	The actual assertion of the host machine's interrupt signal.
Download	The process of transferring transmit data from system memory to the adapter.
Upload	The process of transferring receive data from the adapter to system memory.
Byte	An 8-bit wide quantity of data.
Word	A 16-bit wide quantity of data (2 bytes).
Double Word (dword)	A 32-bit wide quantity of data (4 bytes).

Typographic Conventions

The following typefaces are used to distinguish between object types.

Object Type	Example	Typeface
Register Name	**RxStatus**	Helvetica bold font; first character and embedded words capitalized
Signal or Register Field Name	*txComplete*	Times italic font, first character lower case, embedded words capitalized
`Code or Command Name`	`RxEnable`	Courier font, first character and embedded words capitalized

Register Definitions Legend

The figure below provides a legend for interpreting the register bit map diagrams.

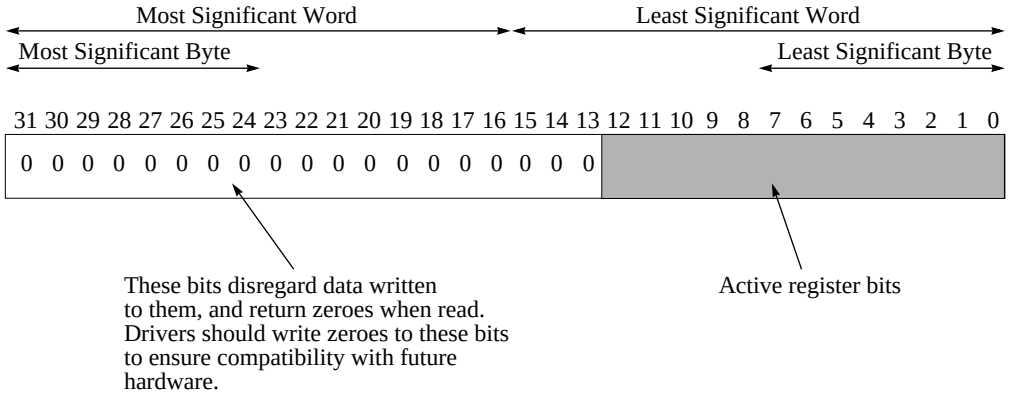


Figure 1-1. Register Bit Map Legend

Chapter 2
PCI/EISA Bus Master Versus PIO

This chapter lists the ways in which PCI/EISA bus master adapters differ programmatically from PIO adapters. It also explains the few differences between PCI and EISA bus master adapters. Refer to the next chapter for an introduction to adapter operation.

Functional Differences from PIO

This section lists the major differences between PCI/EISA bus master and PIO adapters.

Data Transfer Modes

PCI/EISA adapters support fragment-based bus master transfer. The **MasterStatus** register provides transfer-complete bits that can be used for polling. A *transferInt* bit is included in **IntStatus** so that interrupts can be generated upon the completion of bus master transfers.

Extra Register Window

PCI/EISA bus master adapters implement Window 7, which contains the **MasterStatus**, **MasterAddress**, and **MasterLen** registers to support bus master operations.

Statistics Registers

On PCI/EISA bus master adapters, some of the statistics registers have been increased in size to support the higher data rate required by 100 Mbps operation. The registers and their sizes for PIO adapters and bus master adapters are shown below.

Register Name	PIO Adapter Size (bits)	Bus Master Adapter Size (bits)
BytesXmittedOk	16	16
BytesRcvdOk	16	16
FramesDeferred	8	8
FramesRcvdOk	8	10
FramesXmittedOk	8	10
RxOverruns	8	8
LateCollisions	8	8
SingleCollisionFrames	6	8
MultipleCollisions	6	8
SqeErrors	4	4
CarrierLost	4	4

FramesRcvdOk and **FramesXmittedOk** are 10 bits wide, yet reside in 8-bit wide register spaces, so their high-order bits are made visible in a new register called **UpperFramesOk**.

SqeErrors and **CarrierLost** stick at 0xf, to prevent the host from reading a zero from them as they roll over.

All statistics registers have been redesigned so there is no longer any need to disable statistics collection when registers are read. This prevents statistics events from being lost at 100 Mbps.

Large Packet Support

PCI/EISA bus master adapters support "large" packets, up to 4,494 bytes each, including the Frame Check Sequence (FCS). To accommodate this, all threshold registers have been increased to 13 bits wide. Since commands still only have an 11-bit parameter, all threshold-setting commands write the 11-bit parameter into the upper 11 bits of the threshold registers; the lower two bits are always zero. This has the side effect of scaling all threshold values by a factor of four when PIO code is running on PCI/EISA bus master adapters.

Receive frames are truncated at 1,792 bytes or 6 KB, depending upon the value of *allowLargePackets* in **MacControl**. Errors of the type *oversizedFrame* are flagged at either greater than 1,514 bytes or greater than 4,494 bytes, depending upon the value of *allowLargePackets*.

The **RxStatus** register functionality has been split into two registers, **RxStatus** and **RxError**. The error codes from PIO have been replaced with a bit-per-error condition.

Large packet support affects bit-field widths in the following registers: **RxStatus**, **MasterStatus**, **TxStartThresh**, **TxAvailableThresh**, and **RxEarlyThresh.**

Media-Related Functions

Several functions have been added to support the 100 Mbps data rate and extended deference modes.

The **MacControl** register contains bits to allow setting of transmit deference options.

The **PhysicalMgmt** register allows control over the Management Interface portion of the Media Independent Interface.

Station Address Masking Function

The **StationMask** register provides the ability to treat individual bits in **StationAddress** as unimportant when address comparisons on receive frames are performed.

New Registers

New registers added to PCI/EISA bus master adapters versus those supported on PIO adapters are as follows:

StationMask, PhysicalMgmt, MacControl, MasterLen, **MasterAddress**, **MasterStatus, OtherInt**, **ResetOptions, RxError**, **UpperFramesOk, BadSSD**

Commands Not Supported

This generation of adapters does not support the power management commands implemented on PIO adapters.

New Interrupt Bit

The transferInt bit has been added to **IntStatus** to support transfer complete interrupts.

Miscellaneous Details

Full duplex operation is enabled by setting *fullDuplexEnable* in **MacControl**, instead of *externalLoopback* in **NetworkDiagnostic**.

The *busMasterInProgress* bit has been added to **IntStatus**. The *otherInt* and *transferInt* bits in **IntStatus** are swapped relative to PIO adapters.

Differences Between PCI and EISA Bus Master Adapters

ProductId, **AddressConfig**, **ConfigControl**, **EthernetControllerStatus,** and **ResourceConfig** are not supported by PCI adapters. Most of the functions in these registers have been moved to PCI configuration registers. PCI adapters are configured with special registers in the PCI configuration space.

EISA bus master adapters still support **ProductId**, **AddressConfig**, **ConfigControl**, and **ResourceConfig**.

EISA adapters are configured using the registers in Windows 0, which are mapped into the EISA slot-specific I/O space. Refer to Chapter 5 for more detail.

For both EISA and PCI adapters, the Power-on Reset bits (POR) have been moved into the new register **ResetOptions** at Window 3, offset 8. The transceiver select bits, *autoSelect* and *romSize*, have been moved to **InternalConfig**.

PIO/Bus Master Nomenclature

This manual introduces new naming conventions to make names more meaningful. The following table lists the resulting inconsistencies between the PCI/EISA bus master documents and the PIO adapter documents.

Bus Master Adapter Name	PIO Adapter Name	Type
InterruptEnable	**InterruptMask**	Register
IndicationEnable	**ReadZeroMask**	Register
RxData	**RxPioDataRead**	Register
TxData	**TxPioDataWrite**	Register
Frame start header	Transmit preamble	Data structure
hostError	*adapterFailure*	**IntStatus** bit
interruptRequested	*interruptOnSuccessfulTransmissionRequested*	**TxStatus** bit
txFatalError	*txResetRequired*	**NetworkDiagnostic** bit
forcedConfig	TST	**EepromCommand** bit
eepromBusy	EBY	**EepromCommand** bit
eepromAddress/eepromOpcode	EEPROM COMMAND	**EepromCommand** field

Refer to the section “Typographic Conventions” on page 1-3 for an explanation of the meaning of the typefaces used here.

Chapter 3
Adapter Operation

This chapter contains the following PCI/EISA bus master adapter topics:

- Basic operational concepts
- Configuration and initialization
- Frame transmission
- Frame reception
- Interrupts and indications
- Statistics

PCI and EISA bus master adapters share many of the same characteristics. In this manual, differences are clearly indicated as PCI or EISA. If there is no such indication, the information is valid for either type of adapter.

Basic Operational Concepts

This section contains information about the following basic concepts:

- Register windows
- Bit-widths of register accesses
- Command register
- Command summary
- **IntStatus** register
- Optimized adapter operation
- Timer register
- Data transfer modes
- Support for multiple signaling standards
- BIOS ROM

Register Windows

The host interacts with the adapter mostly through I/O mapped registers. The I/O registers are grouped into eight 16-byte “windows.”

The adapter occupies 32 bytes in the host computer’s I/O space. The first 16 bytes look essentially like the first-generation PIO adapter programming model. At any given time, one of eight possible register windows is visible in this space.

The upper 16 bytes are a fixed window into the Window 1 registers. This fixed window is provided as an aid to writing drivers to operate in multiprocessor environments. When the driver code is split among multiple processors, the critical-path data transfer code always has access to the key data transfer registers.

A register’s location is specified by its window number and its offset within the window. For instance, information about transmit frames is available in the **TxStatus** register at Window 1, offset b and Window 7. Some registers appear in more than one window.

In general, registers are grouped together in a window because they are used together to perform a major adapter function. For instance, Window 0 registers are used for adapter configuration, and Window 1 registers are used for PIO data transfer.

The window currently visible in the first 16 bytes of I/O space is changed by issuing a command to the adapter. Commands are described in the section “Command Summary” on page 3-3.

Bit-Widths of Register Accesses

In general, I/O registers must be accessed with instructions that are no larger than the bit-width of that register. For instance, even though the **BytesRcvdOk**, **UpperFramesOk**, and **FramesDeferred** registers all appear in the double word at offset 8 in Window 6, it is not legal to read all three registers with a single 32-bit I/O read instruction.

Additionally, **StationAddress** and **StationMask** must be accessed with no larger than word-wide (16-bit) cycles because of internal architecture limitations.

Some registers cannot be accessed with cycles narrower than the register. These restrictions are detailed in the individual register definitions.

Command Register

Many of the driver’s interactions with the adapter are performed using a command structure. Commands are codes, sometimes including a parameter, written to the adapter to perform some action. For instance, the `RxEnable` command causes the adapter to start accepting receive frames from the medium.

Commands are written to the **Command** register. The write-only **Command** register is unusual in that it appears in every window. It resides at offset e.

Command Summary

The commands are listed here for reference. Refer to the **Command** register definition in Chapter 4, “Register Listings,” for complete definitions of these commands.

`GlobalReset`	Perform an overall reset of adapter
`SelectRegisterWindow`	Change the visible window
`EnableDcConverter`	Enable the 10BASE2 DC-DC converter (10 Mbps only)
`RxDisable`	Disable frame reception
`RxEnable`	Enable frame reception
`RxReset`	Reset the receive logic
`TxDone`	Signal that a transmit frame has been downloaded to the transmit FIFO
`RxDiscard`	Discard the top receive frame from the adapter
`TxEnable`	Enable frame transmission
`TxDisable`	Disable frame transmission
`TxReset`	Reset the transmit logic
`RequestInterrupt`	Cause the adapter to generate an interrupt
`AcknowledgeInterrupt`	Acknowledge active interrupts
`SetInterruptEnable`	Set the value of the **InterruptEnable** register
`SetIndicationEnable`	Set the value of the **IndicationEnable** register
`SetRxFilter`	Set the value of the **RxFilter** register
`SetRxEarlyThresh`	Set the value of the **RxEarlyThresh** register
`SetTxAvailableThresh`	Set the value of the **TxAvailableThresh** register
`SetTxStartThresh`	Set the value of the **TxStartThresh** register
`StartDma`	Start a bus master data transfer operation
`StatisticsEnable`	Enable collection of statistics
`StatisticsDisable`	Disable statistics collection
`DisableDcConverter`	Disable the 10BASE2 DC-DC converter (10 Mbps only)

IntStatus Register

The read-only **IntStatus** register occupies the same location as the **Command** register: offset e in every window.

IntStatus is used by a driver to determine the sources of interrupts on the adapter, and to determine which window is currently visible. **IntStatus** also includes a bit to indicate when a command issued to the **Command** register is in the process of being executed.

Optimized Adapter Operations

This specification describes several mechanisms by which adapter operations are optimized. An optimization mechanism involves generating an interrupt or starting a process before it would normally occur, to gain some benefit from parallel operation.

Optimization mechanisms involve the use of a threshold register, whose value specifies when the early interrupt or process start should occur. For instance, the **TxStartThresh** register specifies how many bytes of a frame must be downloaded to the adapter before the frame can begin transmission.

Timer Register

The **Timer** register performs interrupt latency measurements to support some of the optimization mechanisms on the adapter. The timer is started when the adapter asserts an interrupt on the bus. Refer to the **Timer** register definition in Chapter 4, “Register Listings,” for complete details.

Data Transfer Modes

Two frame-data transfer modes are supported by PCI/EISA adapters. A driver can use any convenient combination of the transfer modes to perform frame data transfers. The data transfer modes (programmed I/O and bus master) are described briefly in the following paragraphs.

Programmed I/O

Programmed I/O (PIO) is the “base” data transfer mode, supported by all PIO and bus master adapters. Under PIO, a 32-bit wide register in Window 1 is used as a write-only port for transmit data, and as a read-only port for receive data.

PIO data transfers can be of byte, word, or double-word width, and can be at any byte alignment. Prior to performing PIO reads, a driver must first check that there is valid data in the receive FIFO by reading **RxStatus**. Before performing PIO writes, a driver must verify that space exists in the transmit FIFO by reading **TxFree**.

Refer to the sections “Frame Transmission” on page 3-10 and “Frame Reception” on page 3-14 for more details about PIO data transfers.

Bus Master

PCI/EISA adapters support fragment-based bus master transfers. This feature allows burst data transfer at four bytes for every system clock, yielding a maximum raw data transfer rate of over 100 MBps. Actual aggregate data transfer rates will be lower because of on-board RAM contention and contention with other system masters.

Fragment bus master data transfers are controlled using registers located in Window 7. The length and starting address for a data fragment transfer are programmed into **MasterLen** and **MasterAddress**, respectively. A command is then issued to the adapter to cause it to initiate the bus master transfer. Bus master transfers can be of any length and begin on any byte boundary.

Status information for bus master transfers can be read from the **MasterStatus** register.

When a fragment bus master transfer is complete, the adapter sets the appropriate completion bit in **MasterStatus**. The adapter can also be programmed to generate an interrupt upon completion of a bus master operation.

It is expected that bus master transfers will be intermixed with PIO transfers to move packets in the most efficient way. Small fragments will probably be more efficiently moved with PIO because of the overhead involved in setting up a bus master transfer. Host software should measure the overhead to initiate a bus master transfer at initialization time to determine the break-even point between PIO and bus master transfers.

Refer to the sections “Frame Transmission” on page 3-10 and “Frame Reception” on page 3-14 for more details about fragment bus master data transfers.

BIOS ROM

Like other PIO and bus master adapters, Fast EtherLink bus master PCI/EISA adapters support an optional BIOS ROM. A variety of PROMs, EEPROMs, and flash ROMs are supported. The EISA adapter BIOS ROM interface is similar to the one found in previous PIO adapters. The PCI specification requires a few differences in the ROM interface compared to that of PIO adapters.

First, the BIOS ROM is configured through the **BiosRomControl** PCI configuration register. The physical size of the ROM can still be programmed into the EEPROM at the time of manufacture and read from there by a driver, but the PCI system power-on self-test (POST) will not be aware of the ROM’s physical size.

Second, PCI requires that ROMs be accessible using byte, word, or double-word cycles. Therefore, a host read of the adapter’s BIOS ROM receives wait-states while four byte-wide ROM accesses occur on the adapter. All write accesses to the ROM (with a flash or EEPROM device installed) must be made via double-word writes to the adapter.

Configuration and Initialization

This section contains information about the following topics that affect configuration and initialization:

- System reset
- Forced configuration
- Global reset
- PCI adapter configuration
- EISA adapter configuration
- Adapter initialization

System Reset

System reset is the assertion of the hardware reset signal on the PCI or EISA bus. System reset causes a complete reset of the adapter, including forcing flip-flops to known values, and loss of any adapter configuration that has been set.

Forced Configuration

Under some circumstances, it is necessary to force a usable configuration into the adapter ASIC without its being able to read the EEPROM. Examples of this are the ASIC IC production test and board-level production test.

The EEPROM data-in pin is sampled on the trailing edge of the system reset pulse. If it is low, the adapter is forced into the following configuration:

- PCI–I/O base address 0x200, I/O target cycles and bus master cycles enabled, memory target cycles disabled, and BIOS ROM disabled.
- EISA–I/O base address z000 (z equals the EISA slot number), card enable on, and BIOS ROM disabled.

Global Reset

A `GlobalReset` command is available for use by the driver software in resetting the adapter. The `GlobalReset` command has a bit mask parameter that allows selective reset of various parts of the adapter. Refer to the **Command** register definition in Chapter 4 for details.

PCI Adapter Configuration

Adapter configuration consists of allocating system resources to the adapter and setting adapter-specific options. This is done by writing values into special PCI configuration and I/O registers. The location of this configuration space in the host processor's address map is system-dependent. Configuration is performed by a POST routine supplied with the computer system.

The registers that are set during configuration are described in the following paragraphs.

PciCommand	Enables adapter operation by allowing it to respond to and generate PCI bus cycles. This register also allows enabling of parity error generation.
IoBaseAddress	Sets the I/O base address for the adapter.
BiosRomControl	Sets the base address and size for an installed expansion ROM, if any.
LatencyTimer	Programs an adapter timer that controls how long the adapter can hold the bus as a bus master.
InterruptLine	Maps the adapter's interrupt request to a specific interrupt line (level) on the system board.
InternalConfig	Selects the media port (transceiver) and local RAM parameters. **InternalConfig** will probably not be written by the system configuration utility, but it will be mapped in the PCI configuration space for possible future use. **InternalConfig** is also mapped into Window 3 of the I/O register space.

Refer to Chapter 5, "Adapter Configuration," for complete details of the configuration registers.

EISA Adapter Configuration

EISA systems dedicate 1 KB of I/O space to each card slot. EISA adapters are configured using a group of configuration registers, which are mapped to certain locations within this 1 KB slot-specific space. Under the EISA configuration scheme, after system reset the system checks each slot for a unique adapter ID code and matches the codes with data stored in the system's nonvolatile RAM. The system uses that data to program the various EISA configuration registers on the adapter. The basic configuration consists of such parameters as interrupt level and BIOS ROM base address.

See Chapter 5, "Adapter Configuration," for more information on EISA configuration.

Adapter Initialization

After the system has performed basic configuration of the adapter, software needs to initialize the adapter, which means setting the adapter registers to the desired initial values.

Serial EEPROM

The serial EEPROM is used for nonvolatile storage of such information as the device ID, node address, manufacturing data, default configuration settings, and software information. Some of the EEPROM data is automatically read into the adapter logic after system reset (for example, device ID and configuration defaults), whereas other data (for example, node address and software information) is meant to be read by driver software.

Shortly after system reset, EEPROM control logic reads certain locations from the EEPROM, placing the data into the following host-accessible registers:

PCI EEPROM Location	Register	Space
3	**DeviceId**	PCI configuration
12	**InternalConfig** Low	I/O, PCI configuration
13	**InternalConfig** High	I/O, PCI configuration

EISA EEPROM Location	Register	Space
8	**AddressConfig**	EISA configuration
9	**ResourceConfig**	EISA configuration
3	**ProductId**	EISA configuration
12	**InternalConfig** Low	I/O, EISA configuration
13	**InternalConfig** High	I/O, EISA configuration

Selecting the Media Port

The media port (transceiver) to be used is selected through the *xcvrSelect* field in the **InternalConfig** register. On PCI adapters, **InternalConfig** is mapped into both the PCI configuration and I/O register spaces but will in most cases be written in the I/O space.

Because the value of **InternalConfig** is also stored in the serial EEPROM, it is possible to set the media port once, write the value into EEPROM, and then have the adapter automatically use the stored value when it is powered up.

Alternatively, there is a mechanism for having the driver ignore the stored value for *xcvrSelect* and attempt to set the media port based on which one is currently active. This is called auto select. When the *autoSelect* bit in **InternalConfig** is set, the driver selects each port available on the adapter (see **ResetOptions** later in this chapter) in turn, and attempts to determine which port is connected to the network. If the driver fails to find a connected port, it restores the original value in *xcvrSelect*.

The recommended sequence for determining the active port is as follows (a driver skips the steps corresponding to those media types that are not installed on the adapter):

- 100BASE-TX – *linkBeatDetect* in the **MediaStatus** register indicates an active port.
- 100BASE-FX – *linkBeatDetect* in the **MediaStatus** register indicates an active port.
- MII – the indication of an active port depends upon the type of transceiver that is connected to the MII. Typically, the management interface provided by the **PhysicalMgmt** register is used to check for a device connected to the MII.

- 10BASE-T – *linkBeatDetect* in the **MediaStatus** register indicates an active port.
- 10Mbps AUI (10BASE5) – The driver performs an external loopback to check for an active port. Refer to the section “Network Diagnostic” on page 4-34 for more information on loopback modes.
- 10BASE2 – The driver performs an external loopback to check for an active port. Refer to the section “Network Diagnostic” on page 4-34 for more information on loopback modes.

ResetOptions

The **ResetOptions** register provides a way for driver or configuration software to determine the hardware media options installed on the adapter, and the media operational mode (normal or test).

ResetOptions attains its value upon hardware (system) reset, when certain ASIC pins are sampled and latched.

Setting the RAM Partition

The **InternalConfig** register also contains several fields related to the local packet buffer RAM. Three of the fields (*ramSize*, *ramWidth*, and *ramSpeed*) are fixed for a particular adapter, and are not writable by host software. The *ramPartition* field, however, is set by driver or configuration software to tune the adapter to the particular system environment.

The value of *ramPartition* determines how the local packet buffer RAM is divided between receive and transmit functions. It is expressed in terms of a ratio of receive space to transmit space. For instance, an adapter with 64 KB of local RAM with a *ramPartition* setting of 3:1 would have 48 KB of receive space and 16 KB of transmit space.

Refer to the **InternalConfig** register definition for more details on the RAM parameter fields.

Station Address

The driver is expected to program the adapter’s network address into the **StationAddress** register. The adapter’s network address can be obtained from the appropriate data locations within the EEPROM. The host is, of course, free to program any arbitrary value into **StationAddress**.

Broadcast Address

To have the adapter respond to broadcast frames, frames with a broadcast address (that is, ff:ff:ff:ff:ff:ff) can be received by setting the *receiveBroadcast* bit in **RxFilter**.

Multicast Addresses

The adapters include no support for multicast comparisons. The Fast EtherLink bus master adapters can be configured to accept all multicast frames by setting the *receiveMulticast* bit in **RxFilter**. Any further filtering must be accomplished in software.

Capabilities Word

The Capabilities Word is a 16-bit location in the EEPROM that specifies the capabilities of the adapter. Refer to the section “EEPROM Data Format” on page 5-8 for more details.

Frame Transmission

This section contains an overview of the frame transmission process.

Frame Transmission Model

The frame transmission mechanism is modeled as a logical FIFO. Data to be transmitted is transferred into the FIFO by the system interface and is removed from the FIFO by the network interface.

Within the FIFO, frames are delimited by a 32-bit frame start header (FSH), which includes the length of the frame that follows. The frame data consists of the destination address field through the info field. The adapter normally generates a CRC and inserts it into the frame check sequence (FCS) field automatically, although it is an option to disable this and supply the FCS along with the frame data.

Transmit data is moved into the FIFO using programmed I/O or bus master transfers. The methods used to download the data are transparent to the FIFO.

Frame transmission involves downloading (writing) the FSH and the transmit frame to the FIFO, issuing a `TxDone` command to inform the adapter that the frame is complete, and possibly responding to a transmit complete indication.

Transmit Data Writes

The transmit FSH and frame data is written to the transmit FIFO using combinations of two data transfer modes: programmed I/O and bus master.

Programmed I/O

The first method is through I/O writes to the **TxData** register. This register is a special-purpose window into the transmit FIFO.

The data can be written to **TxData** as bytes, words, or double-words and can be aligned to any byte lane.

Prior to writing transmit frame data to the adapter, the host must verify that sufficient space remains in the transmit FIFO. This is determined by reading **TxFree**. The value returned by **TxFree** indicates the number of bytes of free space within the transmit FIFO.

If the driver determines that the free space is smaller than the frame to be transmitted, then the driver should issue the `SetTxAvailableThresh` command and await an interrupt. Upon responding to the *txAvailable* interrupt, sufficient room within the transmit FIFO is ensured.

Bus Master

In this second method, the host places data to be transmitted in a contiguous block of system memory. The starting address of the fragment buffer is written to the **MasterAddress** register, and the byte length of the buffer is written to **MasterLen** (the start address may have any byte alignment). The adapter's `StartDma` command is then issued to cause the adapter to perform the bus master transfer.

Unlike PIO, the adapter paces the data transfers so that no transmit FIFO overrun occurs during bus master transfers. It is not necessary to check the value of **TxFree** before a bus master download operation is started.

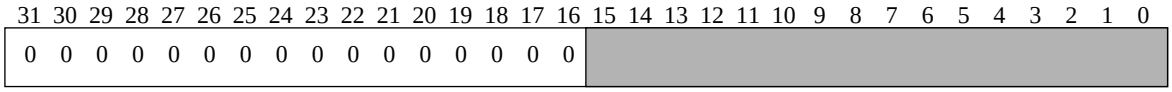
When the bus master transfer is complete, the adapter sets the *masterDownload* bit in the **MasterStatus** register. The adapter can also be programmed to generate an interrupt upon completion of a bus master download operation.

While a bus master download operation is in progress, it is an error to write to the **TxData** register, since this would insert data at unpredictable places in the transmit frame. The value in **TxFree** is also unreliable during bus master download operations.

Frame Start Header

The first 32 bits of data transferred to the transmit FIFO are interpreted to be a frame start header. The frame start header contains frame length and control information for the frame.

The format of the frame start header is as follows:



txIndicate [15]: This bit is set if the driver desires a *txComplete* indication upon completion of transmission of this frame. If this bit is cleared, no indication of transmit completion is given by the adapter.

reserved [14]: This bit is reserved. A zero should be written here.

crcAppendDisable [13]: The driver sets this bit to inhibit the adapter from appending a CRC to the end of this frame. When *crcAppendDisable* is set, it is expected that the frame's CRC will be supplied as part of the data downloaded to the FIFO. An exception to this is a transmit underrun, in which case a guaranteed-bad CRC is appended to the frame.

When this bit is cleared, the adapter computes and appends CRCs for transmit frames.

An adapter indicates its support of *crcAppendDisable* through the Capabilities Word.

txLength [12:0]: This field contains the length of the frame to follow. When the driver writes data to the FIFO, it must pad the data to the next dword boundary (or use the `TxDone` command). The value in *txLength* must match the number of actual frame bytes and must not include any pad bytes.

Completing a Transmit Frame Download

The downloading of a transmit frame can be "completed" in one of two ways. In either case, the goal in completing a frame is to ensure that the frame is properly delimited and readied for transmission in the transmit FIFO.

The two ways to complete a frame download are:

- Padding to a double-word boundary
- Issuing a TxDone command

The need to properly complete a transmit frame applies equally whether the frame is being moved with PIO, bus mastering, or a combination of the two methods.

Padding to a Dword Boundary

To pad to a dword boundary, the driver must write sufficient bytes after the last data bytes of the packet to make the total number of bytes written for the packet be an even multiple of four.

For example, if a 61-byte packet is to be transmitted, and the FSH and 61 bytes of data have been downloaded, then three more bytes must be written, either as three-byte writes or a word write plus a byte write.[1] With bus mastering, this can also be accomplished by padding out the length of the final fragment transfer to include the extra bytes. Note that these pad bytes are important in getting the packet into the transmit FIFO and hence available for transmission. The transmit packet could underrun after all data bytes have been written if too much time elapses before the pad bytes are written.

Issuing a TxDone Command

The TxDone command is used when it is inefficient for the driver to pad the final write to the transmit FIFO. This might occur when the driver is using a bus master transfer to move the last fragment of a frame, and it is desired to write the fragment length as specified from upper layers of software directly into the **MasterLen** register.

When TxDone is used, the total number of frame bytes written to the FIFO must exactly match the frame length specified in the frame start header. After the last write operation to the FIFO, the driver issues TxDone to inform the adapter logic that the frame is complete.

Padding to Minimum Frame Length

The adapter automatically appends the appropriate number of arbitrary data bytes to the end of the frame's data field to pad the frame to 60 bytes in length. The frame length value written as part of the frame start header does not reflect the padded frame length, but rather the number of bytes supplied by the driver to the adapter.

Initiating Frame Transmission

The adapter initiates frame transmission as soon as either the entire frame is resident on the adapter or the number of bytes that are resident is greater than the value in **TxStartThresh**.

1. This example does not represent a particularly efficient way to move this 61-byte packet into the FIFO. A more efficient method would be to move the FSH and packet data in 17 dword write instructions: one dword for the FSH and 16 dwords for the packet data plus pad bytes.

Transmission Completion

As soon as the adapter has completed its attempt(s) to transmit a frame, it can post the frame's transmit status. Status is not posted unless either an error occurred during frame transmission or the *txIndicate* bit in the frame start header was set.

Assuming that *txIndicate* for the frame is set and the interrupt and indication masks are appropriately configured, then the adapter generates an interrupt on the host's bus.

The host's initial reaction to an interrupt should be to read the **IntStatus** register to determine the cause of the interrupt. The *txComplete* bit in **IntStatus** is set.

Once the device driver has determined that the adapter has completed its attempt to transmit a frame, it may examine **TxStatus** to determine the outcome of the transmission.

Updating the Status

When the device driver has checked the outcome of a transmission in **TxStatus**, writing an arbitrary value to **TxStatus** causes **TxStatus** to advance to the results of the next transmit frame, if one exists.

Multiple Transmit Completions

If more than one frame has pending transmit status, *txComplete* remains set after the write to **TxStatus**, until the status of all completed transmissions has been read. If there are no more frames that have completed transmission, then *txComplete* is cleared and the other bits in **TxStatus** are undefined.

After servicing a *txComplete* interrupt and writing to **TxStatus**, the driver should test to see whether *txComplete* remains asserted. If *txComplete* is not asserted, then the driver should return from the interrupt service routine. If, however, *txComplete* is set, then the driver should read the **TxStatus** register for the status of the next completed transmission.

Frame Transmission Errors

Although **TxStatus** is only of interest to the driver when *txComplete* indications are required, an exception to this is created when transmission errors occur. In this situation, a *txComplete* indication is always issued and transmit processes are stopped. When the host responds to the indication, it checks **TxStatus** for the cause of the error. Status for previous, successful transmit frames (that had the *txIndicate* bit set) may need to be read first to bring the error frame's status to the top of the **TxStatus** queue.

When a transmission error occurs, the driver must reenable the transmitter before subsequent transmissions can proceed. Some transmission errors may require the driver to reset the transmit logic to recover from the error.

Optimized Transmit Operations

Early Transmission Start

The adapter can be enabled to begin transmission of a frame onto the network media prior to the transfer of the entire frame into the adapter's transmit FIFO. This is accomplished via **TxStartThresh**.

TxStartThresh specifies the number of bytes of the transmit frame that must reside on the adapter before it can commence with the transmission of the frame. The start of transmission may be delayed by other queued transmissions or by delays in gaining media access.

If this register is set to a value that is greater than the maximum allowed frame length, then the early transmit feature is disabled and the entire transmit frame must reside on the adapter before the adapter will begin to transmit it.

The value for this register must be carefully chosen to optimize performance. If set too low, system latencies or bandwidth limitations may cause the adapter to underrun the network during transmission, causing a partial (bad) frame to be transmitted. (The frame will have a bad CRC appended, guaranteeing its rejection by receiving stations.) If set too high, unnecessary delays are incurred before the start of transmission.

The adapter generates an indication of an underrun via the *txUnderrun* bit in **TxStatus**. A driver responds to a *txUnderrun* error by first waiting for any transmission in progress to finish by polling on the *txInProg* bit in **MediaStatus**. Next, the driver should reset the transmit process, including any active bus master download, using `TxReset`. The frame that experienced the underrun can then be resubmitted to the adapter for transmission.

When a *txUnderrun* indication does occur, the driver should increase the value in **TxStartThresh**. Further *txUnderrun* indications should cause the driver to continue to increase **TxStartThresh**. If **TxStartThresh** is eventually greater than the maximum allowable frame length, then the early transmit start feature is disabled.

Frame Reception

This section contains an overview of the frame reception process.

Frame Reception Model

The frame reception mechanism is modeled as a logical FIFO. Data to be received is transferred into the FIFO by the network interface, and removed from the FIFO by the system interface.

The frame data placed in the FIFO consists of the Destination Address field through the information field. The adapter normally strips the FCS automatically, although it is an option to disable this feature, causing the adapter to supply the FCS along with the frame data.

Receive data is moved out of the FIFO using PIO or bus master transfers. The methods used to upload the data are transparent to the FIFO.

The adapter receives all frames that meet the filtering criteria established by bits in **RxFilter** and by the address stored in **StationAddress**.

Top Frame

The concept of the “top receive frame” must be understood before any explanation of receive operations. The top frame is the oldest frame in the receive FIFO, and hence the frame for which status is available in **RxStatus**. The top frame can be either a frame that is currently being received off the network media or one that has already been completely received.

Normal Frame Reception

In the nonoptimized condition, the driver is alerted to the presence of a receive frame by an interrupt on the host’s bus. The *rxComplete* bit in **IntStatus** indicates to the driver that an entire frame has been received and is available in the receive FIFO. The received frame is available for examination upon receipt of the *rxComplete* interrupt.

Receive Data Reads

The receive frame data is removed from the receive FIFO using combinations of two data transfer modes available on the adapter: programmed I/O and bus master.

Programmed I/O

The first method is through I/O reads of the **RxData** register. **RxData** is a 32-bit window into the adapter’s receive FIFO. Sequential reads of bytes, words, and double words are allowed on any combination of contiguous byte lanes.

Prior to reading data from **RxData**, a driver must determine that there are bytes available to be read in the receive FIFO. This is done by checking the *rxBytes* field in the **RxStatus** register.

In general, it is illegal to read bytes from **RxData** beyond the last byte of the top frame. An important exception is to read extra bytes (up to three) within the last double word of the frame. This allows an entire frame to be read with 32-bit I/O cycles.

Bus Master

In this second method, the host writes the address of a fragment buffer into the **MasterAddress** register, and the byte length of the buffer into the **MasterLen** register (the start address may have any byte alignment). The adapter’s `StartDma` command is then issued to cause the adapter to perform the bus master transfer.

Unlike PIO, the adapter paces the data transfers such that no receive FIFO underrun occurs during bus master uploads. It is not necessary to check the value of *rxBytes* before a bus master download operation is started.

When the bus master transfer has been completed, the adapter sets the *masterUpload* bit in the **MasterStatus** register. The adapter can also be programmed to generate an interrupt upon completion of a bus master upload operation.

While a bus master upload operation is in progress, it is an error to read from the **RxData** register, as this would remove data from the uploaded receive frame at unpredictable places. It is also illegal to issue an `RxDiscard` command while a bus master operation is in progress. Also, the values in *rxBytes* are unreliable during bus master upload operations.

RxStatus and RxError

Associated with the top receive frame is information related to the condition of the received frame and the number of bytes of the top frame that remain in the receive FIFO. This information is available via **RxStatus** and **RxError**.

RxStatus provides status information for receive frames and the number of bytes remaining in the FIFO for receive frames. Receive overruns, framing errors, CRC errors, and the oversized and runt frame conditions are the error conditions reported in **RxError**.

The length value returned by **RxStatus** represents the number of valid data bytes that remain to be transferred from the FIFO. Data bytes beyond the end of the frame are undefined and in some cases are illegal to read.

Discarding the Top Frame

Issuing the `RxDiscard` command causes the adapter to make the next receive frame the top frame. Until `RxDiscard` is issued, the status and data of the subsequent frame are unavailable to the host. Once `RxDiscard` is issued, the status of the top frame are discarded and cannot be recovered.

Queued Receives

If the driver is unable to keep up with the adapter's rate of frame reception, receive frames are queued up within the adapter's on-board receive FIFO.

If one or more received frames are queued on the adapter when `RxDiscard` is issued by the driver, an *rxComplete* indication occurs, informing the driver that the receive FIFO contains a new valid receive frame.

Receive Frame Size Limits

The adapter truncates any received frame to a length of 1,792 or 6,144 bytes, depending upon the value programmed into the *allowLargePackets* bit in **MacControl**.

Also, depending upon the value in *allowLargePackets*, the adapter generates an *oversizeFrame* error for receive frames greater than 1,518 bytes or 4,494 bytes, including the FCS field.

Optimized Frame Reception

Early Receive Indications

A method is available for early indications of received frames. This involves setting a threshold relative to the start of a frame.

RxEarlyThresh is used to program an indication threshold relative to the start of the incoming frame. The first byte of the destination address is considered to be byte 1.

RxEarlyThresh is set using the `SetRxEarlyThresh` command. The current threshold setting can be read in the **RxEarlyThresh** register.

As soon as the number of bytes that have been received is greater than the value in **RxEarlyThresh**, the adapter generates an interrupt to the host (assuming the *rxEarly* indication and interrupt bits are not masked). The *rxEarly* indication only occurs when the frame being received is the top frame. In other words, the *rxEarly* indication only occurs if the frame being received can be transferred by the host during reception. The **RxEarlyThresh** mechanism causes one *rxEarly* indication per frame unless it is retriggered.

An *rxEarly* indication occurs whenever the **RxEarlyThresh** threshold has been exceeded and the frame being received is the top frame. These two conditions can be met in either order. In other words, it is reasonable to expect that issuing the `RxDiscard` command may cause an *rxEarly* indication by making a frame that is in the process of being received the top frame.

The driver can program any value into **RxEarlyThresh**, but setting **RxEarlyThresh** to less than 8 causes the adapter to interpret the value as 8, so as to allow the adapter to perform destination address filtering before generating an *rxEarly* indication.

RxEarlyThresh also involves the concept of frame "visibility." The value programmed into **RxEarlyThresh** determines how many bytes of a frame must be received before information about the frame is made visible in **RxStatus**. Frames become visible when **min (60, RxEarlyThresh)** bytes are received (that is, frames become visible after 60 bytes or when the number of bytes set in **RxEarlyThresh** has been received).

For bus master transfers, the value in **RxEarlyThresh** also determines how many bytes of a frame must be received before upload transfers for the frame are allowed to begin.

Setting **RxEarlyThresh** to a value that is too low may cause the host to process an excessive number of collision fragments. Setting **RxEarlyThresh** to a value that is too high introduces unnecessary delays in the system's receive response sequence.

If **RxEarlyThresh** is set to a value that is greater than the length of the received frame, then an *rxComplete* interrupt occurs at the completion of frame reception rather than an *rxEarly* interrupt.

If the host system is particularly slow in responding to an *rxEarly* interrupt, then it is entirely likely that the frame will have been completely received by the time the driver examines the adapter. In this case, *rxEarly* is overridden by *rxComplete*. The *rxEarly* and *rxComplete* interrupts are mutually exclusive. Because *rxEarly* "goes away" when *rxComplete* becomes set, *rxComplete* should only be disabled if *rxEarly* is also disabled. This prevents spurious interrupts.

rxEarly is meant to be usable as a retriggerable interrupt. In other words, it is legal for the driver to respond to an *rxEarly* interrupt because of a value set in **RxEarlyThresh** and then reprogram **RxEarlyThresh** to a larger value so that a subsequent interrupt is generated within the same receive frame. If a new value is set in **RxEarlyThresh** while a frame is being received from the medium, then an *rxEarly* indication is generated as soon as the *rxEarly* threshold is crossed, or it is generated immediately if the threshold has already been crossed.

Discarding a Frame During Reception

By issuing the `RxDiscard` command, the driver can discard a frame while it is being received. A frame discarded in this way does not generate an *rxComplete* interrupt.

Interrupts and Indications

Interrupts are used to assert a signal to the host that asynchronous activities deserve the host's attention, whereas indications are read from a status register.

Interrupts Versus Indications

There is an important distinction between interrupts and indications. An interrupt results in the assertion of the interrupt signal on the host bus. An indication is merely a bit set in the **IntStatus** register that can be read by the driver. All of the sources of interrupts on the adapter can be used as indications or as indications and interrupts.

Determining the Cause of an Interrupt

When responding to an interrupt, the host reads **IntStatus** to determine the cause of the interrupt.

IntStatus

Seven bits in this register define the source of the interrupt. The least significant bit of **IntStatus**, *interruptLatch*, is always set whenever any of the interrupts are asserted. This prevents spurious interrupts on the host bus. The *interruptLatch* interrupt must be explicitly acknowledged using the `AcknowledgeInterrupt` command.

Interrupt Acknowledgment

The host acknowledges interrupts by carrying out interrupt-specific actions. These actions are as follows:

interruptLatch	acknowledged by `AcknowledgeInterrupt` command
txComplete	acknowledged by writing to **TxStatus**
rxComplete	acknowledged by `RxDiscard` command
rxEarly	acknowledged by `AcknowledgeInterrupt` command
intRequested	acknowledged by `AcknowledgeInterrupt` command
hostError	acknowledged by issuing the appropriate resets
updateStats	acknowledged by reading one or more statistics registers
txAvailable	acknowledged by `AcknowledgeInterrupt` command
otherInt	acknowledged by clearing interrupt sources in **OtherInt**
transferInt	acknowledged by clearing interrupt sources in **MasterStatus**

Interrupt and Indication Enable Mechanisms

Figure 3-1 illustrates the relationship between interrupts, indications, and their respective enable mechanisms.

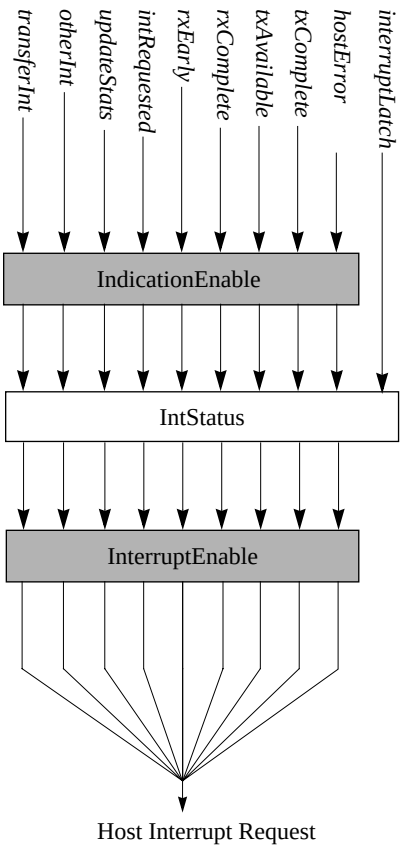


Figure 3-1. Interrupt and Indication Enable Mechanisms

An interrupt is an asynchronous indication that an event has taken place on the adapter that requires the attention of the host system. The host, however, may not respond quickly, or may not respond at all to certain events. The architecture provides a flexible scheme for allowing each type of event to be assigned the level of urgency sought by the host.

The shaded boxes in the figure above represent enable mechanisms. The white box is the **IntStatus** register, which the host uses to view the various indication bits.

Enable mechanisms have an immediate effect on indications and interrupts. In other words, if a particular interrupt is pending and the host clears its enable bit in **IndicationEnable**, the indication, though still pending, would appear as a zero in **IntStatus** and would no longer contribute to the assertion of the interrupt line on the host bus. Conversely, if the pending indication were to be enabled (by setting its enable bit), the indication would become set in the appropriate registers, and the interrupt signal on the host bus would be asserted by the adapter.

Masking prevents an interrupt or indication from acknowledging the interrupting event.

Before exiting the interrupt service routine, the host should recheck the **IntStatus** register to determine whether any further asynchronous events have occurred.

Statistics

The architecture includes specifications for 12 statistics counters of various widths. The gathering of statistics is enabled by issuing the `StatisticsEnable` command. When enabled, the statistics counters advance as the corresponding events occur. No host intervention is required to facilitate this counting.

Reading a statistics register clears it. It is not necessary to disable statistics collection while reading the statistics registers. It is legal to do so, but disabling statistics collection may result in missed statistical events.

Whenever any of the statistics registers reaches the half-way point of its count, it generates an *updateStatistics* interrupt. Reading a statistics register clears it.

Writing a value to a statistics register adds that value to the register. This is useful in diagnostics and IC production tests.

Reading all of the statistics will acknowledge the *updateStatistics* interrupt.

Transmit Statistics

FramesXmittedOk — The number of frames of all types transmitted without errors. Loss of carrier and absence of an expected SQE are not considered to be errors by this statistic.

BytesXmittedOk — A byte total for all frames transmitted without error.

FramesDeferred — If the transmission of a frame had to defer to network traffic, the event is recorded in this statistic. A single frame may defer more than once as a result of collisions; each deferral would be counted.

SingleCollisionFrames — Frames that are transmitted without errors after one and only one collision (including late collisions) are counted by this register.

MultipleCollisions — All frames transmitted without error after experiencing from 2 through 15 collisions (including late collisions) are counted here.

LateCollisions — Every occurrence of a late collision (there could be more that one per frame transmitted) is counted by this statistic.

CarrierLost — Frames that were transmitted without error but experienced a loss of carrier are counted by this statistic.

SqeErrors — If the adapter is configured to expect an SQE pulse after each transmission and did not receive such a pulse, the event is counted here.

Receive Statistics

FramesRcvdOk

Frames of all types that are received without error are counted here.

BytesRcvdOk

A byte total for all frames received without error. A frame's bytes are included in this count if the frame is received without errors and the frame is completely moved into the receive FIFO before `RxDiscard` is issued.

RxOverruns

This statistic is a count of *rxOverrun* errors. Only frames that are actually seen as overruns by the host are included in this count. Frames that are completely ignored by the adapter because of a full receive FIFO are not included.

BadSSD

A count of frames received with a bad start-of-stream delimiter. This statistic applies only to 100BASE-TX or 100BASE-FX operation.

Chapter 4
Register Listings

I/O Model

PCI/EISA bus master adapters present and map a set of registers to the host CPU I/O space. Since there are far more registers than is advisable to I/O map directly in DOS systems, the registers are broken into several groups that are made available to the host through 32 bytes worth of I/O space.

The lower 16 bytes of the I/O space look essentially like the PIO redundant model used in previous 3Com adapters. At any given time, one of eight possible register banks (windows) is visible in this space.

The upper 16 bytes are a fixed window into the Window 1 register set.

PCI and EISA bus master adapters share many of the same characteristics. In this manual, differences are clearly indicated as “PCI Only” or “EISA Only.” If there is no such indication, the information is valid for either type of adapter.

PCI/EISA bus master adapter register windows are shown below.

byte 3	byte 2	byte 1	byte 0	Offset	Window
IntStatus/Command		MasterStatus		c	7
TxStatus	Timer	RxStatus		8	
MasterLen		TBD	RxError	4	
MasterAddress				0	

IntStatus/Command		BytesXmittedOk		c	6
BytesRcvdOk		UpperFramesOk	FramesDeferred	8	
FramesRcvdOk	FramesXmittedOk	RxOverruns	LateCollisions	4	
SingleCollFrames	MultipleColl's	SqeErrors	CarrierLost	0	

IntStatus/Command		IndicationEnable		c	5
InterruptEnable		RxFilter		8	
RxEarlyThresh		Reserved		4	
TxAvailableThresh		TxStartThresh		0	

IntStatus/Command		Reserved	BadSSD	c	4
MediaStatus		PhysicalMgmt		8	
NetworkDiagnostic		FifoDiagnostic		4	
VcoDiagnostic		Reserved		2	2

IntStatus/Command		TxFree*		c	3
RxFree		ResetOptions		8	
MacControl		RomControl (EISA)	OtherInt	4	
InternalConfig				0	

IntStatus/Command				c	2
StationMask (High)		StationMask (Mid)		8	
StationMask (Low)		StationAddress (High)		4	
StationAddress (Mid)		StationAddress (Low)		0	

IntStatus/Command		TxFree		c	1
TxStatus	Timer	RxStatus		8	
Reserved		Reserved	RxError	4	
TxData/RxData				0	

IntStatus/Command		EepromData		c	0
EepromCommand		ResourceConfig (EISA)		8	
AddressConfig (EISA)		ConfigControl (EISA)		4	
ProductId (EISA)		ManufacturerID (EISA)		0	

* TxFree in Window 3 is for diagnostic purposes only22
and should **not** be used by drivers.

Register Definitions

This section gives precise definitions of the PCI bus master adapter registers. The figure below provides a legend for interpreting the register bit map diagrams.

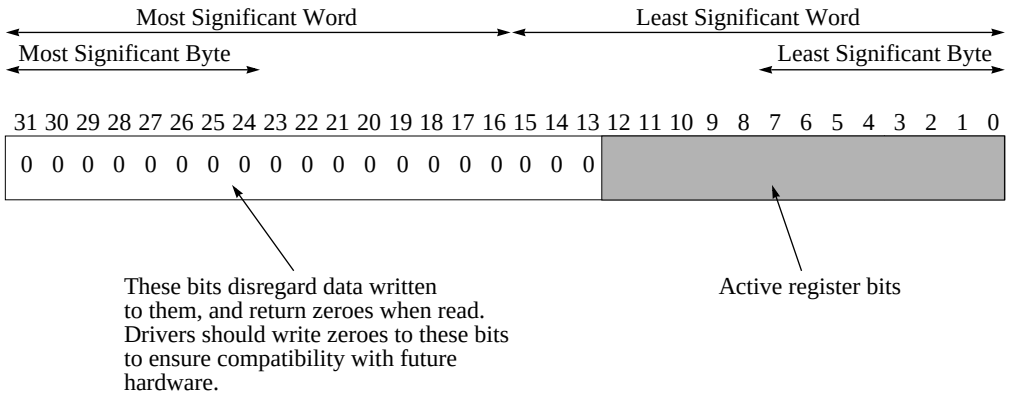


Figure 4-1. Register Bit Map Legend

AddressConfig (EISA Only)

Synopsis Provides access to the BIOS ROM configuration information and a copy of the *xcvrSelect* field from the **InternalConfig** register.

Type Read/Write

Size 16 bits

Window 0

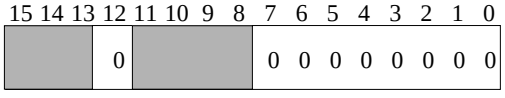
Offset 6

Definition

AddressConfig provides access to the BIOS ROM configuration information.

The *romBase* field of **AddressConfig** is loaded with the value from the corresponding bits in EEPROM word 8, after system reset. See the section “EEPROM Data Format” on page 5-8 for the default values.

The register format is as follows:



romBase [11:8]: This field defines the base address of the BIOS ROM, according to the following ROM Configuration Table (the *romSize* field is located in the **InternalConfig** register).

Table 4-1. ROM Configuration Table

romSize	*romBase*	Address Window
00 (8 K)	0000	Boot ROM Disabled
	0001	0xc2000 to 0xc3fff
	0010	0xc4000 to 0xc5fff
	0011	0xc6000 to 0xc7fff
	0100	0xc8000 to 0xc9fff
	0101	0xca000 to 0xcbfff
	0110	0xcc000 to 0xcdfff
	0111	0xce000 to 0xcffff
	1000	0xd0000 to 0xd1fff
	1001	0xd2000 to 0xd3fff
	1010	0xd4000 to 0xd5fff
	1011	0xd6000 to 0xd7fff
	1100	0xd8000 to 0xd9fff
	1101	0xda000 to 0xdbfff
	1110	0xdc000 to 0xddfff
	1111	0xde000 to 0xdffff
01 (16 K)	0000	Boot ROM Disabled
or	0001	0xc0000 to 0xc3fff
10 (32 K)	001x	0xc4000 to 0xc7fff
or	010x	0xc8000 to 0xcbfff
11 (64 K)	011x	0xcc000 to 0xcffff
	100x	0xd0000 to 0xd3fff
	101x	0xd4000 to 0xd7fff
	110x	0xd8000 to 0xdbfff
	111x	0xdc000 to 0xdffff

The *romSize* for 16, 32, and 64 K has the same *romBase* decode, since only 16 K is mapped at a time. The active 16 K is defined by the *romPage* field within the **RomControl** register.

xcvrSelect [15:13]: This read-only field is a copy of the corresponding read/write bits in the **InternalConfig** register, reflecting the selected transceiver type. Refer to the **InternalConfig** register for their definition.

BadSSD

Synopsis Counts the number of receive frames that have a corrupted start-of-stream delimiter.

Type Read/Write

Size 8 bits

Window 4

Offset c

Definition

This statistic counts the number of packets that are received with a bad start-of-stream delimiter. This statistic is only valid for operating in 100BASE-TX or 100BASE-FX operation.

This is an 8-bit counter that wraps around to zero after reaching 0xff. An *updateStatistics* interrupt occurs after the counter has counted through 0x80. Reading this statistic clears it. Therefore, this statistic must be read as an 8-bit quantity. The `StatisticsEnable` command must have been issued for this register to count events.

BytesRcvdOk

Synopsis Counts the total number of bytes for frames that are received without error.

Type Read/Write

Size 16 bits

Window 6

Offset a

Definition

This statistic counts the number of bytes that are received successfully. For the purposes of this statistic, a successfully received frame is one that is completely moved into the receive FIFO before being discarded by `RxDiscard`.

This is a 16-bit counter and wraps around to zero after reaching 0xffff. An *updateStatistics* interrupt occurs after the counter has counted through 0x8000. Reading this statistic clears it. Therefore, this statistic must be read as a 16-bit quantity. The `StatisticsEnable` command must have been issued for this register to count events.

BytesXmittedOk

Synopsis Counts the total number of bytes for frames that are transmitted without error.

Type Read/Write

Size 16 bits

Window 6

Offset c

Definition

This statistic counts the number of bytes included in frames that are transmitted with no errors reported in **TxStatus**.

This is a 16-bit counter. It wraps around to zero after reaching 0xffff. An *updateStatistics* interrupt occurs after the counter has counted through 0x8000. Reading this statistic clears it. Therefore, this statistic must be read as a 16-bit quantity. The `StatisticsEnable` command must have been issued for this register to count events.

CarrierLost

Synopsis Counts the number of frames experiencing loss of carrier during transmission.

Type Read/Write

Size 8 bits

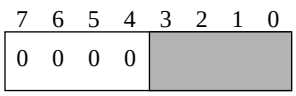
Window 6

Offset 0

Definition

This statistic register counts the number of frames that experience at least one loss of carrier during transmission.

The following bit format is defined for this register:



Carrier sense is not monitored for the purpose of this statistic until after the preamble and start-of-frame delimiter. This is a 4-bit counter that sticks at 0x0f. An *updateStats* indication occurs after the counter has counted through 0x08.

Reading this statistic clears it. The `StatisticsEnable` command must have been issued for this register to count events.

Command

Synopsis Allows for commands to be issued to the adapter.

Type Write only

Size 16 bits

Window All

Offset e

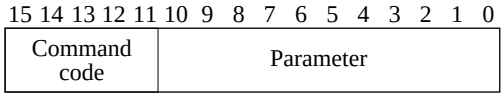
Definition

The **Command** register is used to issue commands of various types to the adapter. Commands may or may not contain parameters. Most commands execute in less time than it takes for the host system to perform a subsequent read or write operation and are considered able to execute in zero time. Those commands that take a nonzero amount of time to execute are so identified in their respective definitions.

All commands must be issued as a single write to **Command**. If the command being issued has Xs occupying bits 7 through 0, then a write to bits 15 through 8 only (offset 0xf) may be used. If any of the least significant eight bits of the command word are defined, then a single 16-bit write must be used. **IntStatus** (read only) is colocated with **Command**.

Command Register Format

The figure below describes the **Command** register format.



A sample command is as follows:

`GlobalReset(0000 0000 •••• ••••)*`

In the command definitions that follow, the 16-bit value in parentheses following the command name is the value that the adapter expects to be written to **Command** to carry out the desired operation. The most-significant five bits make up the command code, and the remaining bits are the parameter. Bit positions occupied by “X” indicate that the value for the corresponding bit does not matter. However, for future hardware compatibility it is recommended that zeroes be written to these positions. Bit positions occupied by “•” indicate those bit positions that are to be filled by the parameter associated with the command.

An asterisk (*) following commands indicates that those commands may not complete execution before the next command can be issued to the adapter. For these commands the host must ensure that the *commandInProgress* bit in **IntStatus** is a zero before taking any further action with the adapter.

Supported Commands

The following commands are supported:

`GlobalReset` (`0000 0000 •••• ••••`) *

This command causes various parts of the adapter to be reset, depending upon the value passed in the 8-bit parameter. When the parameter is cleared, `GlobalReset` has the same effect as a power-up reset except that the adapter's configuration is unaffected. Setting individual bits in the parameter causes the reset to be masked to specific modules, as described in the following paragraphs.

tpAuiReset	[0]: When set, masks reset to the 10BASE-T and 10BASE5 transceivers.
endecReset	[1]: When set, masks reset to the internal Ethernet encoder/decoder.
networkReset	[2]: When set, masks reset to the network interface logic, including the CSMA/CD core.
fifoReset	[3]: When set, masks reset to the FIFO control logic.
aismReset	[4]: When set, masks reset to the autoinitialize state machine logic. If this bit is not set, the EEPROM data is reloaded.
hostReset	[5]: When set, masks reset to the bus interface logic. If *hostReset* is not set, all registers related to the host interface (**IntStatus**, **InterruptEnable**, and **IndicationEnable**, but not the configuration registers) will be cleared.
dmaReset	[6]: When set, masks reset to the bus master logic, including **MasterStatus**, **MasterAddress,** and **MasterLen**.
vcoReset	[7]: When set, masks reset to the on-board 10 Mbps VCO.

Except for the adapter configuration aspects that are handled by the Power-on Self-Test (POST) routines executed by the host, the adapter must be reinitialized after a `GlobalReset` unless *aismReset* is set.

The registers in the PCI configuration space are not reset by the `GlobalReset` command, except those registers that are aliased from registers in the I/O space—**InternalConfig**, **ResetOptions**, and **EepromData**.

Because the adapter's serial EEPROM may need to be read as part of the reset process, this operation can take as long as 1 ms to complete. The *commandInProgress* bit in **IntStatus** must be polled to ensure that the command has been completed.

`SelectRegisterWindow` (`0000 1000 0000 0•••`)

This command causes the specified register bank to become visible in the 16-byte register window.

Register bank zero is the default bank upon system reset.

`EnableDcConverter` (`0001 0XXX XXXX XXXX`)

This command enables (applies power to) the DC-DC converter that drives an on-board 10BASE2 transceiver. This command affects only 10BASE2 operation and should be used only when an adapter is so configured.

After the adapter is powered up or when it experiences a hardware reset, this command must be issued before the 10BASE2 port can be used to transmit or receive frames. The driver should wait at least 800 µs after issuing this command before attempting to transmit or receive frames. The adapter's **Timer** register can be used to time this.

`RxDisable` (`0001 1XXX XXXX XXXX`)

Issuing this command prevents the adapter from receiving any further frames. Any frame that is in the process of being received when this command is issued is not affected. `RxDisable` has no effect on the contents of the receive FIFO or on any receive status or statistics.

`RxEnable` (`0010 0XXX XXXX XXXX`)

This command enables the adapter to receive frames that meet the address filtering requirements currently in use. If this command is issued while a frame is currently active on the network, the adapter begins reception at the beginning of the next frame.

The adapter comes out of reset with the receiver disabled. `RxEnable` must be issued to allow the adapter to receive frames. Either `RxDisable` or `RxReset` can be used to disable receive operations.

`RxReset` (`0010 1000 0•00 ••••`)*

This command resets the receive logic throughout the adapter.

The 5-bit parameter acts as a bit-mask, masking the reset to various portions of the receive logic, as follows:

tpAuiRxReset	[0]: When set, masks reset to the 10BASE-T and 10BASE5 transceiver receive logic.
endecRxReset	[1]: When set, masks reset to the internal Ethernet encoder/decoder receive logic.
networkRxReset	[2]: When set, masks reset to the network interface receive logic, including the CSMA/CD core. If not set, the receiver is disabled, and **RxFilter** is cleared.
fifoRxReset	[3]: When set, masks reset to the receive FIFO control logic. If this bit is not set, the receive FIFO contents are flushed and **RxEarlyThresh** is set to its reset (disabled) state.
dmaRxReset	[6]: When set, masks reset to the bus master logic, including **MasterStatus**, **MasterAddress,** and **MasterLen**.

This command should not be used after initialization except to recover from receive errors such as a receive FIFO underrun.

`TxDone` (`0011 1XXX XXXX XXXX`)*

Issuing this command signals to the adapter that the data which has been downloaded to the transmit FIFO is a complete frame. Issuing `TxDone` has the effect of "flushing" any remaining data in host interface registers into the FIFO.

Refer to the section "Frame Transmission" on page 3-10 for information on when to use `TxDone`.

`RxDiscard` (`0100 0XXX XXXX XXXX`) *

This command causes the top receive frame to be discarded. An `RxDiscard` must be issued for every receive frame. If the top frame has been completely read out of the receive FIFO, then `RxDiscard` causes the **RxStatus** register to reflect the status of the next frame in sequence. If the top frame is still being received or has been only partially read, `RxDiscard` causes the remainder of the frame to be discarded and the data and status of the next frame to become available via **RxData** and **RxStatus**, respectively.

`TxEnable` (`0100 1XXX XXXX XXXX`)

This command enables the adapter to transmit frames. The adapter comes out of reset with the transmitter disabled. This command must be issued before attempts are made to transmit frames. The transmitter can be disabled through the use of the `TxDisable` or `TxReset` commands or by a transmitter error such as a transmit FIFO overrun.

`TxDisable` (`0101 0XXX XXXX XXXX`)

This command disables the adapter's transmitter after the completion of the transmission attempt of any frame currently being transmitted. If additional frames are queued up in the transmit FIFO, they are not transmitted, nor are they discarded. If the transmitter is again enabled, frames in the transmit FIFO are transmitted.

`TxReset` (`0101 1000 0•00 ••••`) *

This command resets the transmitter logic throughout the adapter. `TxReset` is required after a transmit underrun or jabber error.

The low-order bits in the parameter act as a bit-mask, masking the reset to various portions of the transmit logic, as follows:

tpAuiTxReset	[0]: When set, masks reset to the 10BASE-T and AUI (10BASE5) transceiver transmit logic.
endecTxReset	[1]: When set, masks reset to the internal Ethernet encoder/decoder transmit logic.
networkTxReset	[2]: When set, masks reset to the network interface transmit logic, including the CSMA/CD core. If not set, the transmitter is disabled and the **TxStatus** stack is cleared.
fifoTxReset	[3]: When set, masks reset to the transmit FIFO control logic. If this bit is not set, the transmit FIFO is flushed, and **TxStartThresh** and **TxAvailableThresh** are forced to their reset (disabled) state.
dmaTxReset	[6]: When set, masks reset to the bus master logic. If this bit is not set, all bus master logic is reset, including **MasterStatus**, **MasterAddress**, and **MasterLen**.

`RequestInterrupt` (`0110 0XXX XXXX XXXX`)

This command sets the *intRequested* bit in **IntStatus** (if so enabled) and causes an interrupt to the host (also if so enabled).

`AcknowledgeInterrupt` (`0110 1X•• X••X •XX•`)

The `AcknowledgeInterrupt` command resets selected interrupt indications. When issued, the indications that correspond to bits set to one in the parameter field are cleared.

Several of the interrupt types must be acknowledged by means that are unique to the interrupt type. These means are defined in the **IntStatus** register definition.

Attempting to acknowledge an indication that is not active has no effect.

The following bits are defined for this command:

interruptLatchAck	Bit 0
txAvailableAck	Bit 3
rxEarlyAck	Bit 5
intRequestedAck	Bit 6

`SetInterruptEnable` (`0111 0X•• •••• •••X`)

The parameter of this command becomes the value held in the **InterruptEnable** register. Each bit corresponds to an individual interrupt source. Refer to the **IntStatus** register definition for the map of the interrupt bits. Interrupts disabled via this command do not cause an interrupt to the host but may still be set in **IntStatus**. The **InterruptEnable** register is cleared upon adapter reset. Bit 0 of this register does not matter because the *interruptLatch* bit is always enabled.

`SetIndicationEnable` (`0111 1X•• •••• •••X`)

The parameter of this command becomes the value held in the **IndicationEnable** register. Each bit corresponds to an individual indication source. Refer to the **IntStatus** register definition for the map of the indication bits. Indications disabled via this command do not cause an interrupt to the host, nor are they visible in **IntStatus**. The **IndicationEnable** register is cleared upon system reset. Bit 0 of this register is not important because the *interruptLatch* bit is always enabled.

`SetRxFilter` (`1000 0000 0000 ••••`)

This command is used to define the value of the **RxFilter** register. The four active parameter bits in this command may be used in any combination and are defined as follows:

Parameter	Addresses Enabled
XXX1	Individual (must match station address)
XX1X	All multicast (including broadcast)
X1XX	Broadcast
1XXX	All (promiscuous)

The effect of each bit is additive. That is, a 0011_2 pattern would enable individually addressed frames that match the adapter's **StationAddress** as well as all multicast frames. Setting bit 3 (promiscuous) would override bits 2 through 0.

`SetRxEarlyThresh` (`1000 1••• •••• ••••`)

This command is used to set **RxEarlyThresh** to the desired value. The parameter is written into bits [12:2] of **RxEarlyThresh**, and bits [1:0] are cleared.

When the number of bytes received for a frame is greater than the value stored in **RxEarlyThresh**, the *rxEarly* indication is asserted.

For more information on the operation of *rxEarly* interrupts, see the **RxEarlyThresh** register description.

`SetTxAvailableThresh` (`1001 0••• •••• ••••`)

This command is used to set **TxAvailableThresh** to the desired value. The parameter is written into bits [12:2] of **TxAvailableThresh**, and bits [1:0] are cleared.

The *txAvailable* indication is asserted when the number of bytes of free space in the transmit FIFO is greater than the value of **TxAvailableThresh**.

`SetTxStartThresh` (`1001 1••• •••• ••••`)

This command is used to establish the value of **TxStartThresh**. The parameter is written into bits [12:2] of **TxStartThresh**, and bits [1:0] are cleared.

The adapter begins transmission attempts for a frame as soon as the number of bytes downloaded to the transmit FIFO is greater than the value in **TxStartThresh**. If the frame being transmitted is shorter than **TxStartThresh**, then transmit attempts start as soon as the entire frame has been downloaded.

`StartDma` (`1010 0XXX XXXX XX••`)

This command is used to initiate bus master operations (the name is something of a misnomer for bus master adapters). The parameter specifies the direction of the transfer to be started.

Parameter Value	Direction
00	Upload
01	Download
1X	Reserved

`StatisticsEnable` (`1010 1XXX XXXX XXXX`)

This command enables the adapter's statistics counters. Upon power-up, statistics counting is disabled. This command must be issued to enable the counting of statistics events.

`StatisticsDisable` (`1011 0XXX XXXX XXXX`)

To disable the counting of statists events, use this command. It halts the statistics counters. Disabling the counters does not alter their values.

`DisableDcConverter` (`1011 1XXX XXXX XXXX`)

This command disables the DC-DC converter that drives an on-board 10BASE2 transceiver. This command affects only 10BASE2 operation and should be used only when an adapter is so configured.

The driver should wait at least 800 µs after issuing this command before attempting to use an AUI (10BASE5) interface. The adapter's **Timer** register can be used to time this.

Reserved Command Codes

The following command codes are reserved and are ignored by the adapter.

00110	Reserved.
11000	Was `SetTxReclaimThresh` in PIO-only adapters.
11011	`PowerUp` command on adapters that support power management.
11100	`PowerDownFull` command on adapters that support power management.
11101	`PowerAuto` command on adapters that support power management.

ConfigControl (EISA Only)

Synopsis Provides the EISA Expansion Board Control register (card enable) function required by the EISA specification as well as providing the settings of the power-on-reset options.

Type Read/Write

Size 16 bits

Window 0

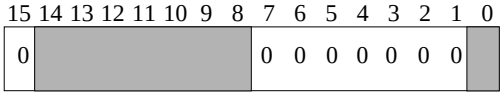
Offset 4

Definition

The lower byte of **ConfigControl** provides the Expansion Board Control register function required in EISA machines. **ConfigControl** is unconditionally decoded at address `0zC84` for this purpose. The EISA system BIOS enables the adapter by setting the *cardEnable* bit in this register.

The upper byte of **ConfigControl** reflects the setting of the power-on reset options, which specify the hardware configuration of the adapter. These bits are read-only and are also available in **ResetOptions** in Window 3.

The register format is as follows:



cardEnable [0]: When set, the adapter is enabled. When cleared, the adapter is disabled.

baseT4Available [8]: When set, indicates that a 100BASE-T4 is available on the adapter through the Media Independent Interface (MII).

baseTXAvailable [9]: When set, indicates that a 100BASE-TX PHY is available on the adapter.

baseFXAvailable [10]: When set, indicates that a 100BASE-FX PHY is available on the adapter.

10bTAvailable [11]: When set, indicates that a 10BASE-T encoder/decoder and transceiver are available on the adapter.

coaxAvailable [12]: When set, indicates that a 10BASE2 coaxial transceiver is available on the adapter.

auiAvailable [13]: When set, indicates that a 10 Mbps AUI connector is available on the adapter.

miiConnector [14]: When set, indicates that an MII-based connector is available on the adapter.

EepromCommand

Synopsis Allows commands to be issued to the serial EEPROM controller.

Type Read/Write

Size 16 bits

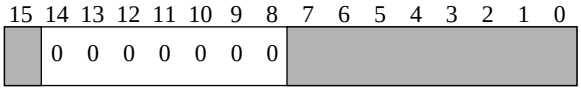
Window 0

Offset a

Definition

This command provides the host with a method for controlling the adapter's serial EEPROM. Individual 16-bit word locations within the EEPROM may be written, read, or erased. Also, the EEPROM's `WriteEnable`, `WriteDisable`, `EraseAll`, and `WriteAll` commands can be issued.

The following fields within **EepromCommand** are used:



The bit fields within **EepromCommand** are defined as follows:

eepromAddress [5:0]: These six read/write bits identify one of the 64 sixteen-bit words to be the target for the `ReadRegister`, `WriteRegister`, and `EraseRegister` commands. Bits 5 and 4 are further defined to identify an individual command among the following group of four subcommands:

Subopcode	Subcommand
00	WriteDisable (60 µs)
01	WriteAll (11 ms)
10	EraseAll (11 ms)
11	WriteEnable (60 µs)

The definition of bits 5 and 4 is valid when *eepromOpcode* in bits 7 and 6 equals 00_2.

eepromOpcode [7:6]: These two read/write bits specify one of three individual commands and a single group of four subcommands. The following table defines the opcodes:

eepromOpcode	**Command**
00	Write Enable/Disable and Write/Erase All subcommands
01	WriteRegister (11 ms)
10	ReadRegister (162 µs)
11	EraseRegister (11 ms)

eepromBusy [15]: This read-only bit is set during the execution of EEPROM commands. Further commands should not be issued to **EepromCommand,** nor should data be read from **EepromData** while this bit is set.

Two-bit opcodes and 6-bit addresses are written to the least significant eight bits of this register to cause the adapter to carry out the desired EEPROM command. If data is to written to the EEPROM, the 16-bit data word must be written to **EepromData** by the host before the associated write command is issued. Similarly, if data is to be read from the EEPROM, the read data is available via **EepromData** 162 µs after the `ReadRegister` command has been issued.

A mechanism within the EEPROM automatically disables writes and erasures to prevent accidental data changes should power be interrupted during a write operation. The EEPROM disables writes and erasures after any write or erase command has been executed. To write or erase a series of locations, the host must issue the `WriteEnable` command before any write or erase command.

The serial EEPROM can only clear bits to zero during a write command and cannot set individual bits to ones. Therefore, an `EraseRegister` or `EraseAll` command must be issued before attempts are made to write data to the EEPROM.

The EEPROM is a particularly slow device. It is important that the host wait until the *eepromBusy* bit is cleared before issuing a command to **EepromCommand**.

A typical write operation would be controlled as follows:

1. Verify *eepromBusy* is cleared.
2. Issue `WriteEnable` command. Opcode = 0011 XXXX2
3. Verify *eepromBusy* is cleared.
4. Issue `EraseRegister` command. Opcode = 11aa $aaaa_2$
5. Verify *eepromBusy* is cleared.
6. Issue `WriteEnable` command. Opcode = 0011 $XXXX_2$
7. Write data pattern to **EepromData**.
8. Verify *eepromBusy* is cleared.
9. Issue `WriteRegister` command. Opcode = 01aa $aaaa_2$

EepromCommand defaults to 0x0000 upon reset.

EepromData

Synopsis	Provides data access for the EEPROM.
Type	Read/Write
Size	16 bits
Window	0
Offset	c

Definition

This register is a 16-bit port for use with the adapter's serial EEPROM. Data read out of the EEPROM can be read by the host from **EepromData** when *eepromBusy* becomes cleared. Data to be written to the EEPROM is written to **EepromData** before the write command is issued to **EepromCommand**.

EepromData is cleared after a system reset.

FifoDiagnostic

Synopsis Provides diagnostic read access to the frame-buffering (FIFO) logic.

Type Read only

Size 16 bits

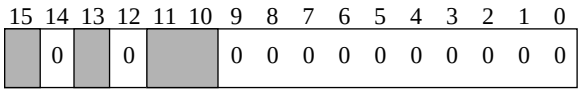
Window 4

Offset 4

Definition

The bits in this register provide various indications of transmit and receive FIFO failures.

The following bit format is defined for this register:



txOverrun [10]: This bit is asserted when the transmit FIFO has run out of room to accept further frame data. The assertion of this bit causes a *hostError* interrupt. A `TxReset` or `GlobalReset` command is required to recover from this condition.

rxOverrun [11]: This bit is set when the receive FIFO is full. It is not necessary for frames to have been discarded for this bit to be set. However, frames received while this bit is set are discarded. This bit is informational only.

No specific action by the host (beyond reading and discarding frames in the receive FIFO) is required. This bit is cleared as soon as the receive FIFO is no longer full.

rxUnderrun [13]: When asserted, this bit causes a *hostError* interrupt that requires either an `RxReset` or a `GlobalReset` command to clear. *rxUnderrun* occurs when the host reads data out of the receive FIFO faster than the network can fill it or faster than the FIFO can supply data to **RxData**, with the result that the host accidentally reads invalid data.

receiving [15]: This bit is set whenever the adapter is receiving a frame in the receive FIFO. No particular action is expected on the part of the host based on the state of this bit.

FramesDeferred

Synopsis Counts the number of transmit frames deferred to network activity.

Type Read/Write

Size 8 bits

Window 6

Offset 8

Definition

This statistic register counts the number of times a transmit frame must defer to network traffic. A single frame may cause multiple deferrals as a result of collisions and retransmissions.

This is an 8-bit counter and wraps around to zero after reaching 0xFF. An *updateStatistics* interrupt occurs after the counter has counted through 0x80. Reading this statistic clears it. The `StatisticsEnabled` command must have been issued for this register to count events.

FramesRcvdOk

Synopsis Counts the number of error-free frames received.

Type Read/Write

Size 8 bits

Window 6

Offset 7

Definition

This statistic register counts the number of frames that are received without error. Frames received with an error are defined as frames in which the *rxOverrun*, *runtFrame*, *alignmentError*, *crcError*, or *oversizedFrame* bit is set in **RxError**. This is a 10-bit counter and wraps around to zero after reaching 0x3ff. An *updateStatistics* indication occurs after the counter counts through 0x200.

The low-order eight bits of this register are visible at this location. The upper two bits are visible in **UpperFramesOk**. When **FramesRcvdOk** is read, the value in the upper two bits of the register is latched and made visible in **UpperFramesOk**. This latched value can be read from **UpperFramesOk** at any time until **FramesRcvdOk** is again read. Reading **UpperFramesOk** has no effect on the value seen in **UpperFramesOk**. Refer to the register definition for **UpperFramesOk** for its bit layout.

The `StatisticsEnable` command must have been issued for this register to count events.

FramesXmittedOk

Synopsis Counts the number of error-free frames transmitted.

Type Read/Write

Size 8 bits

Window 6

Offset 6

Definition

This statistic register counts the number of frames that are transmitted without error. Error frames are defined as those for which *maxCollisions*, *txJabber*, or *txUnderrun* is returned in **TxStatus**. This is a 10-bit counter and wraps around to zero after reaching 0x3ff. An *updateStatistics* indication occurs after the counter counts through 0x200.

The low-order eight bits of this register are visible at this location. The upper two bits are visible in **UpperFramesOk**. When **FramesXmittedOk** is read, the value in the upper two bits of the register is latched and made visible in **UpperFramesOk**. This latched value can be read from **UpperFramesOk** at any time until **FramesXmittedOk** is again read. Reading **UpperFramesOk** has no effect on the value seen in **UpperFramesOk**. See the register definition for **UpperFramesOk** for its bit layout.

The `StatisticsEnable` command must have been issued for this register to count events.

IndicationEnable

Synopsis Specifies which bits in **IntStatus** can become set.

Type Read only

Size 16 bits

Window 5

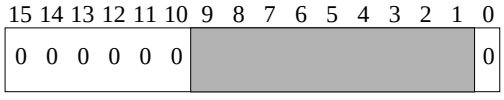
Offset c

Definition

Each bit set in **IndicationEnable** enables the corresponding bit to be set in **IntStatus**. This register is set using the `SetIndicationEnable` command. Refer to the **Command** register description for more details.

IndicationEnable is cleared upon reset.

The register format is as follows:



InternalConfig

Synopsis Allows for setting of adapter-specific configuration.

Type Read/Write

Size 32 bits

Window 3

Offset 0

Definition

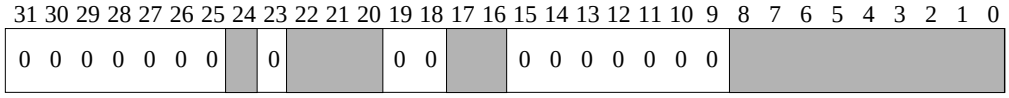
InternalConfig provides a way to set adapter-specific, non-host-related configuration settings. The contents of **InternalConfig** are read from the EEPROM at reset.

The low word of **InternalConfig** (bits [15:0]) contains hardware configuration information that should generally not be changed by software.

The high word (bits [31:16]) contains information that may be changed by installation software to tune the adapter to the system configuration, and a field to select the media port.

InternalConfig contains the value 0102001B hex immediately after reset, but normally it is overwritten with a value loaded from the EEPROM shortly after reset.

The register format is as follows:



ramSize [2:0]: Specifies the size of the packet buffer SRAM installed on the adapter. Depending upon the value of *ramWidth*, as shown below, some values of *ramSize* may be invalid.

0: 8 KB
2: 32 KB
3: 64 KB
4: 128 KB

All other combinations: reserved.

ramWidth [3]: This read-only field specifies the width of the packet buffer RAM.
0: Byte-wide
1: Word-wide

ramSpeed [5:4]: Specifies the number of 25 MHz clock periods required for accesses of the external packet buffer RAM. Fast EtherLink adapters support only 1-clock period accesses, which correspond to the code 01b. This code is always returned here.

romSize [7:6]: Specifies the size of the BIOS ROM installed on the adapter, as defined below.

0: 8 KB
1: 16 KB
2: 32 KB
3: 64 KB

disableBadSsdDet [8]: When set, disables checking for corrupted start-of-stream delimiters (SSD) when operating in 100BASE-TX or 100BASE-FX operation.

ramPartition [17:16]: Specifies how the packet buffer RAM should be divided between the receive FIFO and the transmit FIFO.

0: 5 to 3
1: 3 to 1
2: 1 to 1
3: Reserved

The following shows the valid values of *ramPartition* for the various *ramSize* and *ramWidth* combinations.

ramSize	***ramWidth***	
	Byte	**Word**
8 K	1:1, 3:1, 5:3	–
32 K	1:1	–
64 K	–	1:1, 3:1
128 K	–	1:1

xcvrSelect [22:20]: This read/write field indicates the selected transceiver type:

***xcvrSelect* Value**	**Transceiver Selected**
000	10BASE-T
001	10 Mbps AUI (10BASE5)
010	Reserved
011	10BASE2
100	100BASE-TX
101	100BASE-FX
110	MII
111	Reserved

On a given adapter product, the only legal values for *xcvrSelect* are those that have a corresponding bit set in **ResetOptions**.

After the value of *xcvrSelect* is changed, drivers should issue both `RxReset` and `TxReset`.

autoSelect [24]: When set, indicates that the driver should ignore the value set in *xcvrSelect* and instead auto select the media port at load time. If *autoSelect* is clear, the *xcvrSelect* value is used as is, and the driver configures the adapter accordingly.

Although this bit is read/write, it should be treated as read-only by drivers.

InterruptEnable

Synopsis Specifies which bits in **IntStatus** can generate an interrupt to the host.

Type Read only

Size 16 bits

Window 5

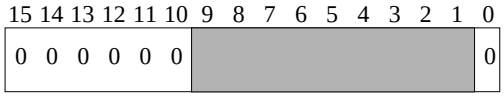
Offset a

Definition

Each bit in **InterruptEnable** is the interrupt enable bit for the corresponding bit in **IntStatus**. Setting a bit in **InterruptEnable** allows that source to generate an interrupt on the bus. This register is set using the `SetInterruptEnable` command. See the **Command** register description for more details.

InterruptEnable is cleared upon reset.

The register format is as follows:



IntStatus

Synopsis Indicates the sources for adapter interrupts, and the number of the visible register window.

Type Read only

Size 16 bits

Window All

Offset e

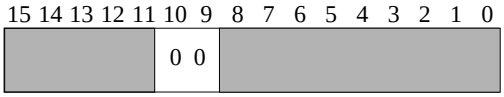
Definition

IntStatus is the main status register for the adapter. It indicates the source of interrupts and indications on the adapter, the completion status of commands issued to the **Command** register, and the current register window visible in the lower part of the I/O space.

Bits 1 through 7 are the interrupt-causing sources for the adapter. These bits can be individually disabled as interrupt sources using the **InterruptEnable** register, and individually forced to read as zero in **IntStatus** using the **IndicationEnable** register.

IntStatus is cleared by a reset.

The register format is as follows:



interruptLatch [0]: Asserted when the adapter is driving the bus interrupt signal. It is a logical OR of the interrupt-causing bits after they have been filtered through the **InterruptEnable** register. *interruptLatch* is acknowledged by issuing the `AcknowledgeInterrupt` command with the *interruptLatchAck* bit set.

hostError [1]: Set when either a transmit overrun or a receive underrun occurs. A transmit overrun occurs when the host writes data to **TxData** when there is no more room for transmit data. A receive underrun occurs when the host reads data from the receive FIFO when there is no valid data to be read. This interrupt is acknowledged by resetting the transmitter or receiver as appropriate.

txComplete [2]: Asserted (1) when a frame whose *txIndicate* bit is set has been successfully transmitted or (2) when any frame experiences a transmission error. This interrupt is acknowledged by writing to **TxStatus** to advance the status FIFO.

txAvailable [3]: Set when the amount of space available in the transmit FIFO exceeds the setting of **TxAvailableThresh**. *txAvailable* is acknowledged by issuing the `AcknowledgeInterrupt` command with the *txAvailableAck* bit set.

rxComplete [4]: Set when one or more entire frames have been received into the receive FIFO. This bit is acknowledged by reading and discarding (using the `RxDiscard` command) all of the complete frames in the FIFO.

rxEarly [5]: Set when the number of bytes of the top frame that have been received is greater than the value of **RxEarlyThresh**. When the top frame has been completely received by the adapter, *rxEarly* is negated and *rxComplete* is asserted (assuming the appropriate masks are clear). *rxEarly* is acknowledged by issuing the `AcknowledgeInterrupt` command with the *rxEarlyAck* bit set.

intRequested [6]: Set by the execution of a `RequestInterrupt` command. It is acknowledged by issuing the `AcknowledgeInterrupt` command with the *intRequestedAck* bit set.

updateStats [7]: Indicates that one or more of the statistics counters is nearing an overflow condition (typically half of its maximum value). Reading all of the statistics acknowledges this bit.

A driver should respond to an *updateStats* interrupt by reading all of the statistics. This has the side effect of acknowledging (clearing) *updateStats*.

transferInt [8]: Indicates that a bus master transfer operation has been completed. When *transferInt* is active, the driver should check **MasterStatus** to determine which transfer direction (upload or download) caused the interrupt. This bit is cleared by clearing *masterUpload/masterDownload* in **MasterStatus**.

busMasterInProgress [11]: Indicates that a bus master transfer is in progress. This bit is set when the `StartDma` command is issued and cleared when the transfer is completed.

commandInProgress [12]: Set to indicate that the last command issued is still being executed by the adapter. It need only be checked after a command is issued that takes more than one I/O cycle for completion. No new commands may be issued until *commandInProgress* is negated.

windowNumber [15:13]: Indicates which set of registers is currently visible in the I/O space of the adapter. The *windowNumber* bit is reset after a hardware reset or a `GlobalReset`.

LateCollisions

Synopsis Returns the number of late collisions during transmission attempts.

Type Read/Write

Size 8 bits

Window 6

Offset 4

Definition

This statistic register counts the number of late collisions. Since every transmission attempt is monitored, it is possible to count multiple late collisions per transmit frame.

This is an 8-bit counter and wraps around to zero after reaching 0xff. An *updateStats* indication occurs after the counter counts through 0x80. Reading this statistic clears it. The `StatisticsEnabled` command must have been issued for this register to count events.

MacControl

Synopsis Allows control of parameters related to Media Access Control.

Type Read/Write

Size 16 bits

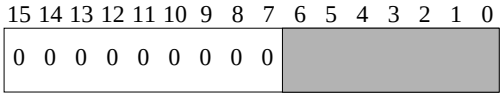
Window 3

Offset 6

Definition

This register provides for setting of MAC-specific parameters. It is cleared upon reset.

The register format is as follows:



deferExtendEnable [0]: Setting this bit enables the special deference mode, in which the time the transmitter defers after a successful transmission is extended to allow other stations collision-free access to the medium. Clearing *deferExtendEnable* causes the adapter to use standard 802.3 deference rules (the values are scaled when operating at 100 Mbps).

deferTimerSelect [4:1]: This field is used to select the amount of time, in addition to the standard Interframe Space (IFS) period, to defer when operating in the special deference modes.

***deferTimerSelect* Value**	**Defer Time**
0	Standard IFS + 0 bit times
1	Standard IFS + 0 bit times
2	Standard IFS + 32 bit times
3	Standard IFS + 64 bit times
4	Standard IFS + 96 bit times
5	Standard IFS + 128 bit times
6	Standard IFS + 160 bit times
7	Standard IFS + 192 bit times
8	Standard IFS + 224 bit times
9	Standard IFS + 256 bit times
A	Standard IFS + 288 bit times
B	Standard IFS + 320 bit times
C	Standard IFS + 352 bit times
D	Standard IFS + 384 bit times
E	Standard IFS + 416 bit times
F	Standard IFS + 448 bit times

When *deferExtendEnable* is clear, the special deference modes are disabled, and the value of *deferTimerSelect* is irrelevant.

fullDuplexEnable [5]: Setting this bit configures the adapter to communicate with the hub in a full-duplex manner. Specifically, it disables transmitter deference to receive traffic, allowing simultaneous receive and transmit traffic.

Setting *fullDuplexEnable* has the side effect of disabling **CarrierLost** statistics collection, since full-duplex operation requires carrier sense to be masked to the transmitter. Software must issue a `TxReset` and an `RxReset` after changing the value of this bit.

allowLargePackets [6]: Setting this bit specifies the frame size at which the *oversizedFrame* error is generated for receive frames.

The table below gives the minimum frame size at which an *oversizedFrame* error will be flagged. The frame size includes the destination and source address, and type/length field, but does not include the FCS field.

***allowLargePackets* Value**	**Minimum *oversizedFrame* Size**
0	1,515 bytes
1	4,491 bytes*

* This value was calculated by taking the maximum FDDI frame size, 4,500 bytes, and subtracting bytes for fields that have no Ethernet equivalent.

ManufacturerId (EISA Only)

Synopsis Returns the EISA manufacturer's ID code.

Type Read only

Size 16 bits

Window 0

Offset 0

Definition

This register eturns the 16-bit manufacturer's code that is hard-wired into the ASIC. The value programmed into word 7 of the EEPROM generally matches **ManufacturerId**, but there is no requirement that this be true.

The value is `0x6d50`, which is the compressed, byte-swapped manufacturer's code assigned to 3Com by the EISA standards body.

MasterAddress

Synopsis Defines the starting address in system memory for a bus master data transfer.

Type Read/Write

Size 32 bits

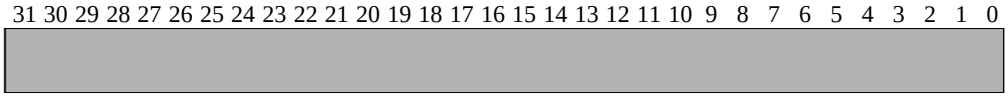
Window 7

Offset 0

Definition

MasterAddress is loaded with the starting address in system memory for a fragment bus master transfer.

The register format is as follows:



For upload operations, **MasterAddress** is the starting address of the buffer to which the fragment of receive data will be transferred. For download operations, **MasterAddress** is the starting address of the buffer to be transferred to the adapter. **MasterAddress** is written before the `StartDma` command is issued.

When read, **MasterAddress** returns the current value of the bus master address counter, which counts up from the original value written to **MasterAddress**. When a bus master transfer is complete, the value in **MasterAddress** is the last byte address of the previous transfer plus 1. Because of this, successive bus master transfers to or from contiguous system memory buffers can be performed without reprogramming **MasterAddress**.

MasterLen

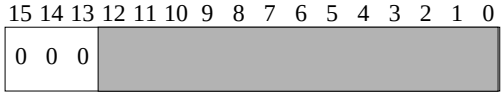
Synopsis	Defines the length of a fragment to be transferred via a bus master operation.
Type	Read/Write
Size	16 bits
Window	7
Offset	6

Definition

MasterLen is loaded with the length of a fragment to be transferred using a bus master operation.

The number of bytes to be transferred is written to **MasterLen** before the `StartDma` command is issued.

The register format is as follows:



When read, **MasterLen** returns the current value of the bus master length counter, which counts down from the original value written to **MasterLen.** As a result, **MasterLen** can be used to determine the number of bytes uploaded for the case in which the value written to **MasterLen** is greater than the remaining number of bytes for a receive frame. When *masterDownload* is set in **MasterStatus**, **MasterLen** correctly reflects the number of untransferred bytes.

MasterStatus

Synopsis Provides status related to bus master operations.

Type Read/Write

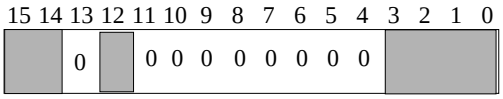
Size 16 bits

Window 7

Offset c

Definition

All bits are set by the adapter and are readable by the software. Some bits can be reset by the software after they have been set by the adapter. The register format is as follows:



masterInProgress [15]: Indicates that a bus master transfer operation is in progress. This bit is unaffected by write operations.

masterUpload [14]: Set when an upload is complete, either because the terminal count was reached or because the end of the receive frame was reached.

When *masterUpload* is set, a *transferInt* interrupt is generated in **IntStatus**, assuming the proper enable bits are set in **InterruptEnable** and **IndicationEnable**.

masterUpload is acknowledged (cleared) by writing a one to it.

masterDownload [12]: Set when an entire download operation is complete.

When *masterDownload* is set, a *transferInt* interrupt is generated in **IntStatus** if the proper enable bits are set in **InterruptEnable** and **IndicationEnable**.

This bit is acknowledged (cleared) by writing a one to it.

targetDisc [3]: Indicates that a target disconnect sequence occurred at some point during a bus master transfer. This bit is informational only, and need not be cleared for subsequent bus master operations to proceed.

targetRetry [2]: Indicates that a target retry sequence occurred at some point during a bus master transfer. This bit is informational only, and need not be cleared for subsequent bus master operations to proceed. This bit is cleared by writing a one to it.

targetAbort [1]: Indicates that a target abort sequence occurred on the last bus master transfer. This bit indicates a fatal error and must be cleared before subsequent bus master operations can proceed. Since a target abort can occur at any point in a transfer, the amount of data written into or read out of the FIFO is indeterminate when this bit is set. A driver will normally perform a DMA reset (`GlobalReset` command with *dmaReset* unmasked) in response to a target abort. This bit is cleared by writing a one to it.

masterAbort [0]: Indicates that the last bus master transfer was aborted because of no response from the addressed slave. This bit must be cleared before a subsequent transfer can occur. This bit is cleared by writing a one to it.

MediaStatus

Synopsis Allows setting of media-specific parameters and provides media-specific status.

Type Read/Write

Size 16 bits

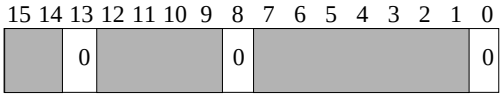
Window 4

Offset a

Definition

This register provides for setting media-specific parameters and for reading media-specific status indications.

The register format is as follows:



dataRate100 [1]: This read-only bit reflects the current operating data rate. When set, the medium is operating at 100 Mbps. When clear, 10 Mbps is the data rate.

For the first generation adapters with PCI bus master architecture, the value of this bit will be determined solely by which media port is selected (*xcvrSelect* in **InternalConfig**).

crcStripDisable [2]: The host asserts this bit when the receive frame's CRC is to be passed to the host as part of the data in the FIFO. The state of *crcStripDisable* does not affect the adapter's checking of the frame's CRC and its posting of CRC error status. *crcStripDisable* is cleared by a system reset.

To avoid confusing the FIFO logic, the value of *crcStripDisable* should only be changed when the receiver is disabled and the receive FIFO is empty.

enableSqeStats	[3]: This read/write bit must be set by the host to enable the **SqeErrors** statistics register to count SQE errors. This bit is normally only set when using an external transceiver across the AUI.
collisionDetect	[4]: This read-only bit provides a real-time indication of the state of the *collisionDetect* signal within the ASIC.
carrierSense	[5]: This read-only bit provides a real-time indication of the state of the *carrierSense* signal within the ASIC.
jabberGuardEnable	[6]: This read/write bit is for use only with the 10BASE-T transceiver. When this bit is set, the adapter automatically shuts down transmissions if it detects that it is not ending transmission normally. *jabberGuardEnable* also enables the automatic reversal of polarity on the receive pair, if required.
linkBeatEnable	[7]: The host sets this read/write bit to require that the adapter detect the presence of the link beat to enable transmission. When *linkBeatEnable* is cleared, the adapter is able to transmit frames with or without detecting the link beat. This bit only works for 10BASE-T or 100BASE TX/FX.
jabberDetect	[9]: This read-only bit is set whenever the adapter senses that it has been transmitting without interruption for much longer than the allowed transmit frame duration. When in this state, the adapter is disabled from further transmissions. The `TxReset` command is required to release the adapter from the jabber detect state.
polarityReversed	[10]: This read-only bit indicates that the twisted-pair transceiver has detected a reversal of polarity on its receive pair. If *jabberGuardEnable* is asserted, then the transceiver automatically corrects the polarity reversal.
linkBeatDetect	[11]: This read-only bit provides a real-time indication of the twisted-pair transceiver link beat status for 10BASE-T, 100BASE-TX, and 100BASE-FX operation. When operating with10BASE-T, this bit reflects the state of the link beat logic. For 100BASE-TX or 100BASE-FX media, this bit reflects the state of the link monitor process. For all speeds, *linkBeatDetect* is forced on whenever *linkBeatEnable* is cleared. For MII operation, this bit is always set.
txInProg	[12]: A real-time indication that a frame is being transmitted. This bit is used by drivers during underrun recovery to delay issuing a `TxReset`.
dcConverterEnabled	[14]: This bit, when set, indicates that the 10BASE2 DC-DC converter has been enabled with the `EnableDcConverter` command.
auiDisable	[15]: This read-only bit is asserted whenever the on-board 10 Mbps transceiver has been selected.

MultipleCollisions

Synopsis Counts the number of transmit frames experiencing at least two collisions.

Type Read/Write

Size 8 bits

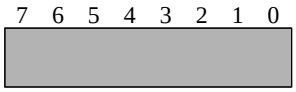
Window 6

Offset 2

Definition

This statistic register counts the number of frames that are transmitted successfully after experiencing anywhere from 2 through 15 collisions or late collisions.

The following bit format is defined for this register:



This is an 8-bit counter and wraps around to zero after reaching 0xff. An *updateStatistics* indication occurs when the counter has counted through 0x80. Reading this statistic has the side effect of clearing it. The `StatisticsEnable` command must have been issued for this counter to be enabled.

Network Diagnostic

Synopsis Provides medium-dependent diagnostic access to the network interface logic, and a few other miscellaneous functions.

Type Read/Write

Size 16 bits

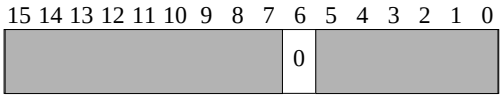
Window 4

Offset 6

Definition

This register provides diagnostic access to the network interface logic in the adapter.

The register format is as follows:



testLowVoltageDetector [0]: Setting this bit tests the low voltage detection circuit, which has the side effect of resetting the adapter. This bit always returns zero.

asicRevision [5:1]: The revision level of the ASIC is reflected in this field. The first operational silicon for the PCI bus master adapter will return 00000b in this field. Significant revisions to the ASIC will cause this field to be incremented.

statisticsEnabled [7]: This read-only bit indicates when the adapter is enabled to count the various statistical events. The value of this bit is affected by the `StatisticsEnable` and `StatisticsDisable` commands.

txFatalError [8]: If a jabber or transmit underrun occurs, this bit is set, indicating that the transmitter needs to be reset with the `TxReset` command.

transmitting [9]: This bit is set whenever the adapter is transmitting or waiting to transmit (deferring).

rxEnabled [10]: This read-only bit is set by the `RxEnable` command and is cleared by the `RxDisable` command, `RxReset` command, or a system reset.

txEnabled [11]: This read-only bit is set by the `TxEnable` command and is cleared by the `TxDisable` command, `TxReset` command, or a system reset.

fifoLoopback [12]: Setting this bit forces data loopback from the transmit FIFO directly into the receive FIFO.

When FIFO loopback mode is used, it is the software's responsibility to ensure that the proper interpacket gap is inserted between frames, to avoid losing data in the receive path. To do this, the software must not load more than one transmit frame into the FIFO at a time.

macLoopback [13]: Setting this bit causes the adapter to loop back transmissions at the output of the media access controller.

endecLoopback [14]: When *endecLoopback* is set, the adapter loops transmissions back to the receiver at the encoder/decoder. When one of the 100BASE-X ports is selected, the loopback path is from the output of the scramble to the input of the descrambler.

externalLoopback [15]: The host asserts this bit to enable the reception of frames transmitted by the adapter. Address filtering criteria must also be met for each frame received.

In 100BASE-FX or 100BASE-TX operation, external loopback occurs on the MAC side of the transceiver chip. For true "on-the-wire" loopback, use a loopback plug, clear all of the loopback bits, and set the *fullDuplexEnable* bit in **MacControl**.

Loopback Mode Notes

`TxReset` and `RxReset` must be issued after the value of any of the loopback bits in **Network Diagnostic** or *fullDuplexEnable* in **MacControl** is changed.

The various loopback modes and their required values for the associated bits in **NetworkDiagnostic**, **MacControl**, and **PhysicalMgmt** are shown below.

Table 4-2. Loopback Modes with Values for NetworkDiagnostic, MacControl, and PhysicalMgmt Registers

Loopback Mode	***fifo Loopback***	***mac Loopback***	***endec Loopback***	***external Loopback***	***full Duplex Enable***	***cat5link TestDefeat***
FIFO Loopback	1	0	0	0	x*	x
MAC Loopback	0	1	0	0	x	x
Encoder/Decoder Loopback	0	0	1	0	x	†
"External" Loopback–100BASE-X‡	0	0	0	1	x	1
True "On-wire" External Loopback–100BASE-X	0	0	0	0	1	1
External Loopback–10BASE-T**	0	0	0	1	x	x
External Loopback–10BASE2††	0	0	0	1	x	x
External Loopback–AUI‡‡	0	0	0	1	x	x
External Loopback–MII***	0	0	0	1	0	0

* x means it does not matter.
† 1 for 100BASE-TX/FX: x for all others.
‡ Loopback through 100BASE-TX/FX transceiver chip, which is not a true "on-the-wire" loopback.
** Requires 10BASE-T loopback plug.
†† Requires loopback plug or coax segment.
‡‡ Loopback type determined by external AUI device.
*** Loopback type controlled by MII device. You may need to enable a loopback mode within the MII device, using the Management Interface.

OtherInt

Synopsis Reports additional adapter interrupt sources.

Type Read/Write

Size 8 bits

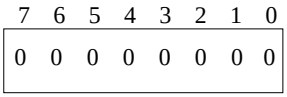
Window 3

Offset 4

Definition

OtherInt is reserved for expansion of the adapter interrupt space in future adapters.

The register format is as follows:



PhysicalMgmt

Synopsis Provides control over various physical layer functions.

Type Read/Write

Size 16 bits

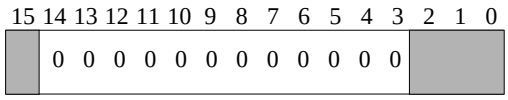
Window 4

Offset 8

Definition

This register allows control over various functions related to the 100BASE-TX implementation.

The register format is as follows:



The following bits control the Media Independent Interface (MII) Management Interface. The Management Interface is a two-wire serial interface connecting the bus master adapter ASIC and any MII-compliant PHY devices residing on the adapter.

Driver software operates the Management Interface by writing and reading bit patterns to the **PhysicalMgmt** register that correspond to the physical waveforms required on the interface signals. For more information on the Management Interface signal protocols, refer to the Reconciliation Sublayer and Media Independent Interface draft supplement to IEEE Std. 802.3.

mgmtClk	[0]: The MII Management Clock bit. This bit directly drives the management clock to the PMD device(s).
mgmtData	[1]: The MII Management Data bit. When the *mgmtDir* bit (below) is set, the value written to this bit is driven onto the MDIO signal. When *mgmtDir* is cleared, data being driven by the PMD can be read from this bit.
mgmtDir	[2]: The MII Data Direction Control bit. Setting this bit causes the ASIC to drive MDIO with the data bit written into *mgmtData*.
cat5LinkTestDefeat	[15]: Setting this bit defeats the link test function in the 100BASE-X reconciliation layer logic. This bit is for diagnostic purposes only; drivers should always write a zero to this bit.

ProductId (EISA Only)

Synopsis	Allows access to the unique adapter identification code.
Type	Read only
Size	16 bits
Window	0
Offset	2

Definition

This register provides access to a unique 16-bit code to identify the adapters. The value in **ProductId** is read from EEPROM word 3 after reset.

The contents of **ProductId** are derived from the 3Com 3C number concatenated with a revision code.

The register format is as follows:

The following **ProductId** values have been defined for EISA bus master adapters:

Fast EtherLink EISA TX (3C597 TX): 5970h
Fast EtherLink EISA T4 (3C597 T4): 5971h
Fast EtherLink EISA MII (3C597 MII): 5972h
EtherLink III Bus Master EISA (3C592): 5920h

The reset-default for **ProductId** is `0x0080`, but this value will be overwritten by the above value in EEPROM before any software has a chance to read the register.

ResetOptions

Synopsis Provides access to the power-on reset options.

Type Read/Write

Size 16 bits

Window 3

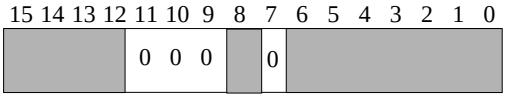
Offset 8

Definition

ResetOptions contains the Power-on Reset (POR) bits, which are latched from certain ASIC pins upon hardware reset.

This register is also visible in the PCI configuration register space.

The register format is as follows:



Bits [8:0] of this register make up the Power-on Reset (POR) bits.

baseT4Available [0]: This read-only bit, when set, indicates that a 100BASE-T4 PHY is available on the adapter through the Media Independent Interface (MII).

baseTXAvailable [1]: This read-only bit, when set, indicates that a 100BASE-TX PHY is available on the adapter.

baseFXAvailable [2]: This read-only bit, when set, indicates that a 100BASE-FX PHY is available on the adapter.

10bTAvailable [3]: This read-only bit, when set, indicates that a 10BASE-T encoder/decoder and transceiver are available on the adapter.

coaxAvailable [4]: This read-only bit, when set, indicates that a 10BASE2 coaxial transceiver is available on the adapter.

auiAvailable [5]: This read-only bit, when set, indicates that a 10 Mbps AUI connector is available on the adapter.

miiConnector [6]: This read-only bit, when set, indicates that an MII-based connector is available on the adapter.

vcoConfig [8]: This read/write bit determines whether the internal VCO or an external one is to be used.

0: External VCO
1: Internal VCO

forcedConfig [12]: When this read/write bit is set, the ASIC is placed in Forced Configuration mode. In this mode, the adapter's PCI configuration register settings are ignored, and the adapter is enabled with the following settings: I/O base address 0x200, I/O target accesses and bus mastering enabled, and memory (BIOS ROM) accesses disabled.

testMode [15:13]: These read-only bits define special test modes, as defined below.

***testMode* Value**	**Operational Mode**
000	10 Mbps AUI Thresholds
001	10 Mbps Receive
010	10 Mbps Transmit
011	10BASE-T Receive Thresholds
100	100BASE-TX/FX Idle
101–110	Reserved
111	Normal Operation

ResourceConfig (EISA Only)

Synopsis Allows interrupt level configuration.

Type Read/Write

Size 16 bits

Window 0

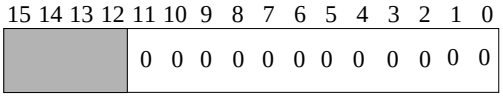
Offset 8

Definition

This register allows setting of the interrupt level configuration.

ResourceConfig is loaded with the value in EEPROM word 9 after system reset. See the section "EEPROM Data Format" on page 5-8 for the default values.

The register format is as follows:



intLevel [15:12]: The 4-bit interrupt-level select code.

Register Value	Bus Interrupt
3xxx	IRQ3
5xxx	IRQ5
7xxx	IRQ7
9xxx	IRQ9
Axxx	IRQ10
Bxxx	IRQ11
Cxxx	IRQ12
Fxxx	IRQ15
All other values	Disabled

RomControl (EISA Only)

Synopsis Controls BIOS ROM functions.

Type Read/Write

Size 8 bits

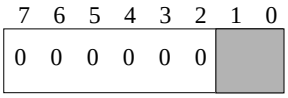
Window 3

Offset 5

Definition

This register is used to control BIOS ROM functions.

The register format is as follows:



romPage [1:0]: This field controls which 16 K page of an installed BIOS ROM is available in the memory window allocated for the adapter. This field is ignored except when ROM sizes of 32 K or 64 K are installed on the adapter.

romPage	**ROM Address Mapped (Hex)**
00	0000-3FFF
01	4000-7FFF
10	8000-DFFF
11	C000-FFFF

reserved [7:2]: These bits are reserved for future expansion. If other bits must be placed here, they should be read-only, so that *romPage* can continue to be written to without the need to first read and mask data.

RxData

Synopsis Provides a PIO port for reading receive data.

Type Read only

Size 32 bits

Window 1

Offset 0

Definition

RxData provides a 32-bit read data port for access to the receive FIFO. Each read that occurs while valid data is available pops the requested number of bytes from the FIFO and presents a new, double-word unit of receive data to **RxData** for a subsequent read.

Data may be read from **RxData** as 8-bit bytes, 16-bit words, or 32-bit double words. These reads of various sizes may be mixed in any combination.

Figure 4-2 shows an example of how this register can be used.

byte 3	byte 2	byte 1	byte 0	entry 1
byte 7	byte 6	byte 5	byte 4	entry 2
byte 11	byte 10	byte 9	byte 8	entry 3
byte 15	byte 14	byte 13	byte 12	entry 4
byte 19	byte 18	byte 17	byte 16	entry 5
byte 23	byte 22	byte 21	byte 20	entry 6
byte 27	byte 26	byte 25	byte 24	entry 7
byte 31	byte 30	byte 29	byte 28	entry 8

Data as originally stored in receive FIFO

byte 3	byte 2	byte 1	byte 0	host read 1
	byte 5	byte 4		host read 2
byte 9	byte 8	byte 7	byte 6	host read 3
byte 11	byte 10			host read 4
byte 14	byte 13	byte 12		host read 5
		byte 16	byte 15	host read 6
			byte 17	host read 7
byte 19	byte 18			host read 8
byte 23	byte 22	byte 21	byte 20	host read 9
	byte 24			host read 10
byte 28	byte 27	byte 26	byte 25	host read 11
		byte 30	byte 29	host read 12
	byte 31			host read 13

Data as read out by the host via **RxData**

Figure 4-2. RxData Example

The upper half of the example in Figure 4-2 shows how the data is arranged in the 32-bit wide receive FIFO immediately after the data has been received. The lower half illustrates how the data is presented in **RxData**. Notice that after each read, regardless of the width of the read, a properly formed 32-bit word is made available in **RxData**.

RxData automatically pads the receive data to allow the use of a 32-bit read operation at the end of the frame even if, for example, only one byte remains to be read. The data that pads the frames is undefined. Reading the pad data is unnecessary for normal operation of the receive FIFO.

After the adapter completes the read of a receive frame, having read either a full or partial frame, it makes the first double-word of the next frame visible in **RxData** by issuing the `RxDiscard` command. `RxDiscard` also updates **RxStatus** to reflect the status of the frame accessible via **RxData**.

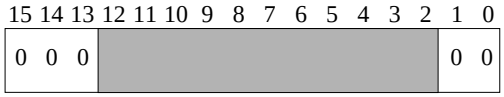
RxEarlyThresh

Synopsis	Returns the value of the **RxEarlyThresh** register.
Type	Read only
Size	16 bits
Window	5
Offset	6

Definition

The value stored in this register defines the number of bytes that must be received before an *rxEarly* indication occurs. The first byte of the destination address is considered to be byte 1.

The register format is as follows:



RxEarlyThresh can be set using the `SetRxEarlyThresh` command.

RxEarlyThresh resets to the value 8188d, which disables the threshold mechanism.

As soon as the number of bytes that have been received is greater than the value in **RxEarlyThresh**, the adapter generates an interrupt to the host (if the *rxEarly* indication and interrupt bits are not masked). The *rxEarly* indication only occurs when the frame being received is the top frame. In other words, the *rxEarly* indication only occurs if the frame being received can be transferred by the host during reception.

The **RxEarlyThresh** mechanism causes one *rxEarly* indication per frame unless it is retriggered. Refer to the explanation of retriggering *rxEarly*, below.

An *rxEarly* indication occurs whenever the **RxEarlyThresh** threshold has been met and the frame being received is the top frame. These two conditions can be met in either order. In other words, it is reasonable to expect that issuing the `RxDiscard` command may cause an *rxEarly* indication by making a frame that is in the process of being received the top frame.

The driver can program any value into **RxEarlyThresh**, but setting **RxEarlyThresh** to less than 8 causes the adapter to interpret the value as 8, to allow the adapter to perform destination address filtering before generating an *rxEarly* indication.

RxEarlyThresh also involves the concept of frame "visibility." The value programmed into **RxEarlyThresh** determines how many bytes of a frame must be received before information about the frame is made visible in **RxStatus**. Frames become visible when **min (60, RxEarlyThresh)** bytes are received (that is, frames become visible after 60 minutes or when the number of bytes set in **RxEarlyThresh** has been received).

For bus master operations, the value in **RxEarlyThresh** also determines how many bytes of a frame must be received before upload transfers for the frame are allowed to begin.

Setting **RxEarlyThresh** to a value that is too low causes the host to respond to the interrupt before the entire receive frame header has been received. Setting **RxEarlyThresh** to a value that is too high introduces unnecessary delays in the system's receive response sequence.

If **RxEarlyThresh** is set to a value that is greater than the length of the received frame, then an *rxComplete* interrupt occurs at the completion of frame reception rather than an *rxEarly* interrupt.

If the host system is particularly slow in responding to an *rxEarly* interrupt, then it is entirely likely that the frame has been completely received by the time the driver examines the adapter. In this case, *rxEarly* is overridden by *rxComplete*. *rxEarly* and *rxComplete* are mutually exclusive. Because *rxEarly* "goes away" when *rxComplete* becomes set, *rxComplete* should only be disabled if *rxEarly* is also disabled. This prevents spurious interrupts.

rxEarly is meant to be usable as a retriggerable interrupt: it is legal for the driver to respond to an *rxEarly* interrupt because of a value set in **RxEarlyThresh** and then reprogram **RxEarlyThresh** to (presumably) a larger value so that a subsequent interrupt is generated within the same receive frame. If a new value is set in **RxEarlyThresh** while a frame is being received from the medium, then an *rxEarly* indication is generated as soon as the *rxEarly* threshold is crossed, or it is generated immediately if the threshold has already been crossed.

RxError

Synopsis Returns the error bits for the top receive frame.

Type Read only

Size 8 bits

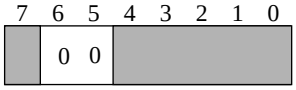
Window 1 and 7

Offset 4

Definition

RxError returns error and informational bits pertaining to the top receive frame.

The following bit format is defined for this register:



rxOverrun [0]: Indicates that software was unable to remove data from the receive FIFO quickly enough, so a data loss condition resulted.

runtFrame [1]: Indicates the frame was less than 60 bytes.

alignmentError [2]: Indicates the frame had an alignment error. This bit applies only to 10 Mbps operation.

crcError [3]: Indicates a CRC error on the frame.

oversizedFrame [4]: Indicates the frame was larger than the maximum allowable size.

The table below gives the minimum frame size at which an *oversizedFrame* error is flagged. The frame size includes the destination and source address and type/length field, but does not include the FCS field.

***allowLargePackets* Value**	**Minimum *oversizedFrame* Size**
0	1,515
1	4,491*

* This value was calculated by taking the maximum FDDI frame size, 4,500 bytes, and subtracting bytes for fields that have no Ethernet equivalent.

dribbleBits [7]: Indicates that the frame had accompanying dribble bits. This bit is informational only, and does not indicate a frame error. This bit applies only to 10 Mbps operation.

RxFilter

Synopsis Defines the types of receive frames that will be accepted.

Type Read only

Size 16 bits

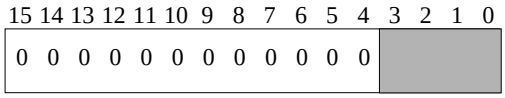
Window 5

Offset 8

Definition

Each bit in **RxFilter**, when set, enables reception of a different type of frame.

The register format is as follows:



receiveIndividual [0]: Setting this bit enables the adapter to receive frames that match the station address set for the adapter.

receiveMulticast [1]: Setting this bit causes the adapter to receive all multicast frames, including broadcast.

receiveBroadcast [2]: Setting this bit causes the adapter to receive all broadcast frames.

receiveAllFrames [3]: Setting this bit causes the adapter to receive all frames promiscuously.

RxFilter is set using the `SetRxFilter` command. It is cleared upon reset.

RxFree

Synopsis Returns the space available in the receive frame buffer area.

Type Read only

Size 16 bits

Window 3

Offset a

Definition

This register provides a real-time indication of the number of bytes of free space that are available in the receive buffer FIFO or memory. If this register returns 0xffff, then at least that many bytes are available. If zero is returned, the receive buffer area is full.

RxFree must be read as a 16-bit quantity to guarantee a valid return value.

RxOverruns

Synopsis Counts the number of frames that cause an *rxOverrun* error.

Type Read/Write

Size 8 bits

Window 6

Offset 5

Definition

This statistic counts the number of frames which should have been received (the destination address matched the filter criteria) but which experienced an *rxOverrun* error because there was not enough FIFO space to hold the frame. This statistic only includes overruns that become apparent to the driver and does not count frames that are completely ignored because the receive FIFO was full at the start of frame reception.

This is an 8-bit counter and wraps around to zero after reaching 0xff. An *updateStatistics* indication occurs after the counter has counted through 0x80. Reading this statistic clears it. The `StatisticsEnable` command must have been issued for this register to count events.

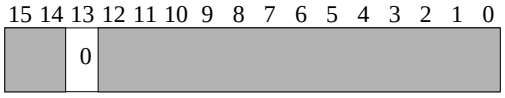
RxStatus

Synopsis	Provides frame status information.
Type	Read only
Size	16 bits
Window	1 and 7
Offset	8

Definition

RxStatus returns the status and the number of bytes residing in the FIFO for the top receive frame. This register may be read multiple times throughout the process of transferring the frame from the adapter to the host. **RxStatus** must be read as a 16-bit quantity to ensure the reliable transfer of the dynamic frame length information from the adapter to the host.

The register format is as follows:



rxBytes [12:0]: This field returns the number of data bytes for the top frame that are available in the receive FIFO.

Since it is possible to read data from the FIFO while the adapter is still receiving the same frame, it is entirely likely that after the adapter has read a block of data from the FIFO, *rxBytes* indicates that a greater number of bytes remain to be read rather than a smaller number.

As the top frame is being received from the medium, bits [1:0] of *rxBytes* are unreliable and should be masked by software. When the frame reception is completed, *rxBytes* then reflects the exact number of bytes remaining in the FIFO.

Frames are padded to dword boundaries within the receive FIFO. It is legal for the driver to read the pad bytes when reading frame data from the FIFO—this improves efficiency of the driver in some cases. It is illegal to read beyond the pad bytes.

rxError [14]: This bit indicates that an error occurred on the top receive frame, as indicated in the **RxError** register. This bit is the logical OR of bits [4:0] in **RxError**.

rxIncomplete [15]: This bit indicates that the frame available in the FIFO and defined here in **RxStatus** is currently being received from the network. When *rxIncomplete* is asserted, *rxError* is zero. Once *rxIncomplete* is false, *rxError* becomes valid.

Issuing the `RxDiscard` command advances **RxStatus** to the status word for the next frame in the receive FIFO, if any.

SingleCollisionFrames

Synopsis Returns the number of frames experiencing a single collision.

Type Read/Write

Size 8 bits

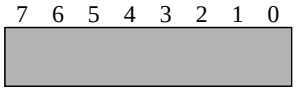
Window 6

Offset 3

Definition

This statistic counts the number of frames that are transmitted without error after experiencing a single collision.

The following bit format is defined for this register:



This is an 8-bit counter and wraps around to zero after reaching 0xff. An *updateStatistics* interrupt occurs after the counter has counted through 0x80. Reading this statistic clears it. The `StatisticsEnable` command must have been issued for this register to count events.

SqeErrors

Synopsis Counts the number of transmit frames that experience SQE errors.

Type Read/Write

Size 8 bits

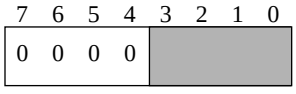
Window 6

Offset 1

Definition

This statistic counts the number of transmit frames that result in an SQE error.

The following bit format is defined for this register:



This is a 4-bit counter and sticks at 0x0f. An *updateStatistics* interrupt occurs after the counter has counted through 0x08. Reading this statistic clears it. The `StatisticsEnable` command must have been issued for this register to count events.

SqeErrors collection can be disabled independently of other statistics if the *enableSqeStats* bit is cleared in **MediaStatus**. Normally, **SqeErrors** would only be enabled when an external transceiver is used over the AUI.

StationAddress

Synopsis Defines the adapter's station address for receive purposes.

Type Read/Write

Size 48 bits

Window 2

Offset 0

Definition

StationAddress is used to define the individual destination address that the adapter responds to when receiving frames. Network addresses are generally specified in the form 00:60:8c:11:22:33, where the bytes appear in the same order as they are transmitted on the network media. To use this particular example address as the adapter's station address, the following writes to **StationAddress** must occur: (1) a write of 0x118c6000 to **StationAddress** + 0; and (2) a write of 0x3322 to **StationAddress** + 4. The writes can be in any order and of any width. The important consideration is that the individual bytes end up in the correct byte lane of the register.

The value programmed into **StationAddress** is *not* inserted into the source address field of frames transmitted by the adapter. The adapter's source address must be specified for every frame as part of the frame contents.

StationMask

Synopsis Defines a mask to apply to the station address register.

Type Read/Write

Size 48 bits

Window 2

Offset 6

Definition

StationMask is a register that allows bits in receive frames to be treated as "don't cares" during individual address matching. Setting a bit in **StationMask** causes the value in the corresponding bit of **StationAddress** to be ignored when the destination address of incoming frames is compared with the adapter's individual address.

Timer

Synopsis Functions as a general-purpose timer.

Type Read-only

Size 8 bits

Window 1

Offset a

Definition

The **Timer** register contains an 8-bit counter that begins counting from zero upon the assertion of the interrupt signal. The host can use this function to make interrupt latency measurements. The counter increments by one every 3.2 µs. When the counter reaches 0xff, it halts. This yields a maximum measurable interrupt latency of 816 µs.

When **Timer** is used to measure interrupt latency, it is suggested that **Timer** be read as late as possible in the interrupt service routine (just before dispatching to handle the interrupt reasons flagged in **IntStatus**) to include the fixed overhead of the interrupt handler itself.

To use **Timer** for general-purpose measurements at driver initialization time, ensure that *interruptLatch* is clear (a pending interrupt would prevent the counter from starting), disable system interrupts, and issue a `RequestInterrupt` command to start the timer.

TxAvailableThresh

Synopsis Returns the value of the **TxAvailableThresh** register.

Type Read only

Size 16 bits

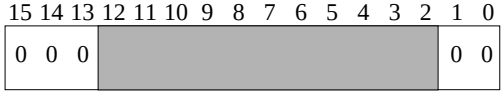
Window 5

Offset 2

Definition

Reading this register returns the value held in **TxAvailableThresh**. The value in **TxAvailableThresh** used to generate a *txAvailable* interrupt is based upon the number of free bytes in the transmit FIFO. A *txAvailable* interrupt is generated when the amount of free space is greater than the value in **TxAvailableThresh**.

The register format is as follows:



The contents of this register reflect the parameter used during the most recent `SetTxAvailableThresh` command. However, when a *txAvailable* interrupt is generated and acknowledged, the process of acknowledging the interrupt will change the value in **TxAvailableThresh** to 8188d, disabling the threshold function. Thus, no more than one *txAvailable* interrupt occurs for each `SetTxAvailableThresh` command issued.

TxData

Synopsis Provides a PIO port for writing transmit data.

Type Write only

Size 32 bits

Window 1

Offset 0

Definition

TxData provides a 32-bit write data port for access to the transmit FIFO. Each write pushes the requested number of bytes onto the FIFO.

Data may be written to **TxData** as 8-bit bytes, 16-bit words, or 32-bit double words. These writes of various sizes may be mixed in any order. Figure 4-3 shows an example of how this register can be used.

byte 3	byte 2	byte 1	byte 0	host write 1
		byte 5	byte 4	host write 2
byte 9	byte 8	byte 7	byte 6	host write 3
	byte 10			host write 4
byte 14	byte 13	byte 12	byte 11	host write 5
byte 16	byte 15			host write 6
			byte 17	host write 7
	byte 19	byte 18		host write 8
byte 23	byte 22	byte 21	byte 20	host write 9
byte 24				host write 10
byte 28	byte 27	byte 26	byte 25	host write 11
		byte 30	byte 29	host write 12
byte 31				host write 13

Data as written to **TxData**

byte 3	byte 2	byte 1	byte 0	entry 1
byte 7	byte 6	byte 5	byte 4	entry 2
byte 11	byte 10	byte 9	byte 8	entry 3
byte 15	byte 14	byte 13	byte 12	entry 4
byte 19	byte 18	byte 17	byte 16	entry 5
byte 23	byte 22	byte 21	byte 20	entry 6
byte 27	byte 26	byte 25	byte 24	entry 7
byte 31	byte 30	byte 29	byte 28	entry 8

Data as stored in the transmit FIFO

Figure 4-3. TxData Example

The upper half of the example in Figure 4-3 illustrates how the data might be written to **TxData**. As long as the host aligns the writes with the least significant byte of the **TxData** register, it is assured of writing contiguous data to the FIFO. The lower half of the above example shows how the data is arranged in the 32-bit wide transmit FIFO immediately after the data has been written to **TxData**.

Padding to Double-Word Boundary

Refer to the section “Frame Transmission” on page 3-10 for an explanation of completing a transmit frame by padding to a double-word boundary.

TxFree

Synopsis Returns the space available in the transmit frame buffer area.

Type Read only

Size 16 bits

Window 1

Offset c

Definition

This register provides a real-time indication of the number of bytes of free space that are available in the transmit buffer FIFO or memory. If this register returns 0xffff, then at least that many bytes are available. If zero is returned, the transmit buffer area is full.

TxFree counts in terms of integral dwords: the low-order two bits of this register are always zero. Hence, **TxFree** is only valid after whole dwords have been moved into the transmit FIFO. The host software is responsible for ensuring that it is always interpreting **TxFree** correctly.

Writing bytes to the transmit FIFO when there is no space causes a transmit overrun condition, which is signaled to the host through the *hostError* indication in **IntStatus**.

TxFree is unreliable while bus master operations are active.

TxStartThresh

Synopsis Provides for an early transmission start based upon the number of frame bytes resident on the adapter.

Type Read only

Size 16 bits

Window 5

Offset 0

Definition

The value in **TxStartThresh** is used to control early frame transmission. Transmission of a frame begins when the number of bytes for the frame resident on the adapter is greater than the value in **TxStartThresh**.

TxStartThresh is set using the `SetTxStartThresh` command.

The feedback to tune this value is the *txUnderrun* status bit in the **TxStatus** register.

The register format is as follows:

This register resets to 8188d, which disables the threshold mechanism.

TxStatus

Synopsis Returns the transmit status for the current transmit frame.

Type Read only

Size 8 bits

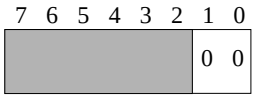
Window 1

Offset b

Definition

This register returns the status of frame transmission attempts. An I/O write of an arbitrary value to this register will advance the status queue to the next transmit status byte.

The register format is as follows:



txStatusOverflow [2]: When set, indicates that the **TxStatus** stack is full and as a result the transmitter has been disabled. Writing **TxStatus** clears this bit, but the transmitter must be reenabled with a `TxEnable` command before transmission may resume.

maxCollisions [3]: When set, the frame was not successfully transmitted because it encountered 16 collisions. Writing **txStatus** clears this bit, but the transmission must be reenabled with a `txEnable` command before transmissions may resume. It is expected that the communications protocols will eventually get around to sending the frame again.

txUnderrun [4]: This bit indicates that the frame experienced an underrun during the transmit process—the host was unable to supply the frame data fast enough to keep up with the network. An underrun will halt the transmitter and the transmit FIFO. The `TxReset` and `TxEnable` commands must be issued before any new frames are submitted to the adapter.

txJabber [5]: This bit is asserted if the adapter determines that it is transmitting for too long. The `TxReset` command is required to recover from this error.

interruptRequested [6]: This bit is asserted if the *txIndicate* bit was set when the 32-bit frame start header was written to the adapter for the frame in question.

txComplete [7]: When this bit is false, then the remainder of the status bits are undefined. When the host chooses to poll this register while waiting for a frame transmission to be completed, then this bit is used to determine whether a frame transmission attempt has been completed. The frame transmission either experienced an error or had the *txIndicate* bit set in the transmit frame descriptor.

UpperFramesOk

Synopsis Makes visible the high-order bits of the **FramesRcvdOk** and **FramesXmittedOk** statistics.

Type Read only

Size 8 bits

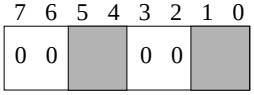
Window 6

Offset 9

Definition

This register allows read access to the high-order bits of the **FramesRcvdOk** and **FramesXmittedOk** statistics.

The register format is as follows:



upperFramesRcvdOk [1:0]: These are the high-order two bits of the **FramesRcvdOk** register. This value is latched whenever **FramesRcvdOk** is read.

upperFramesXmittedOk [5:4]: These are the high-order two bits of the **FramesXmittedOk** register. This value is latched whenever **FramesXmittedOk** is read.

Refer to the **FramesRcvdOk** and **FramesXmittedOk** register definitions for details of how the register values are latched into **UpperFramesOk**.

VcoDiagnostic

Synopsis Allows hardware diagnostic access to the on-board 10 Mbps VCO.

Type Read/Write

Size 16 bits

Window 4

Offset 2

Definition

The register bits are reserved.

Chapter 5
Adapter Configuration

This chapter explains the configuration mechanisms for the PCI and EISA bus master adapters. PCI and EISA bus master adapters share many of the same characteristics. In this manual, differences are clearly indicated as “PCI Only” or “EISA Only,” or each section is labeled for one or the other. If there is no such indication, the information is valid for either type of adapter.

PCI Configuration Overview

PCI bus master adapters use a slot-specific block of configuration registers to perform adapter configuration. The configuration registers are accessed with PCI configuration cycles.

The PCI specification defines two types of configuration cycles. Type 0 cycles are used to configure devices on the local PCI bus. Type 1 cycles are used to pass a configuration request to a PCI bus at a different hierarchical level. PCI configuration cycles are directed at one out of eight possible PCI logical functions within a single physical PCI device.

3Com PCI bus master adapters respond only to Type 0 configuration cycles directed at Function 0: Type 1 cycles and Type 0 cycles directed at functions other than 0 are ignored by the adapter.

Each PCI device is required to decode 256 bytes worth of configuration registers. Of these, the first 64 bytes are predefined by the PCI specification. The remaining registers may be used as needed for PCI device-specific configuration registers.

In PCI configuration cycles, the host system provides a slot-specific decode signal (IDSEL), which informs the adapter that a configuration cycle is in progress. The adapter responds by asserting DEVSEL# and decoding the specific configuration register from the address bus and the byte enable signals. Refer to the PCI BIOS specification for information on generating configuration cycles from driver software.

Figure 5-1 shows the PCI configuration registers used by the adapter.

byte 3	byte 2	byte 1	byte 0	Offset
		EepromData		4c
Reserved		ResetOptions		48
Reserved				44
InternalConfig				40
MaxLat	MinGnt	InterruptPin	InterruptLine	3c
Reserved				38
Reserved				34
BiosRomControl				30
Reserved				2c
Reserved				28
Reserved				24
Reserved				20
Reserved				1c
Reserved				18
Reserved				14
IoBaseAddress				10
Reserved	HeaderType	LatencyTimer	Reserved	0c
ClassCode			Reserved	08
PciStatus		PciCommand		04
DeviceId		VendorId		00

Figure 5-1. PCI Configuration Registers

All spaces marked “Reserved” and all of the locations within the 256 bytes of configuration space that are not shown in Figure 5-1 are not implemented and return zero when read.

PCI Configuration Registers

The following sections describe the various implemented PCI configuration registers.

VendorId

This read-only register contains the unique 16-bit manufacturer’s ID as allocated from the PCI SIG. 3Com’s manufacturer ID is 0x10B7.

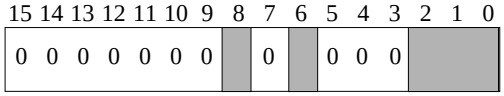
DeviceId

This read-only register contains the vendor-allocated 16-bit device ID for the adapter. This value is read from EEPROM location 03 after system reset.

PciCommand

This read/write register provides coarse control over the adapter's ability to generate and respond to PCI cycles. When a zero is written to this register, the adapter is logically disconnected from the PCI bus, except for configuration cycles.

The register format is as follows:



ioSpace [0]: Setting this bit allows the adapter to respond to I/O space accesses.

memorySpace [1]: Setting this bit (along with *addressDecodeEnable* in **BiosRomControl**) allows the adapter to decode accesses to its BIOS ROM, if one is installed.

busMaster [2]: Setting this bit allows adapters with bus master capability to initiate bus master cycles.

parityErrorResponse [6]: This bit controls how the adapter responds to parity errors. Setting this bit causes the adapter to take its normal action upon detecting a parity error. Clearing this bit causes the adapter to ignore parity errors. This bit is cleared upon system reset.

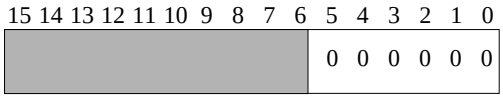
SERREnable [8]: This bit is the enable bit for the SERR# pin driver. A value of zero disables the SERR# driver.

PciStatus

This read/write register is used to record status information for PCI bus events.

Although this register is writable, write operations work in an unusual manner. Read/Write bits in the register can be reset, but not set, by writing to this register. Bits are reset by writing a one to that bit-position.

The register format is as follows:



udfSupported [6]: This read-only bit indicates that the adapter supports the User-Defined Fields format as proposed by the PCI SIG.

fastBackToBack [7]: This read-only bit indicates that the adapter, as a target, supports fast back-to-back transactions, as defined by the criteria in section 3.4.2 of the PCI specification, Rev. 2.0.

dataParityDetected [8]: The adapter sets this bit when, as a master, it detects the PERR# signal asserted and the *parityErrorResponse* bit is set in the **PciCommand** register.

devselTiming	[10:9]: This read-only field is used to encode the slowest time with which the adapter asserts the DEVSEL# signal. The bus master adapter returns 01_2, indicating that it supports “medium” speed DEVSEL# assertion.
signalledTargetAbort	[11]: The adapter asserts this bit when it terminates a bus transaction with target-abort.
receivedTargetAbort	[12]: The adapter asserts this bit when, operating as a bus master, its bus transaction is terminated with target-abort.
receivedMasterAbort	[13]: The adapter asserts this bit when, operating as a bus master, its bus transaction is terminated with master-abort.
signalledSystemError	[14]: This bit is set whenever the adapter asserts SERR#.
detectedParityError	[15]: The adapter asserts this bit when it detects a parity error, regardless of whether parity error handling is enabled.

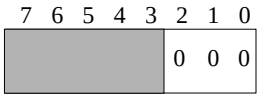
ClassCode

This 24-bit read-only register identifies the general function of the PCI device. The adapter returns either 0x020000 (indicating “Ethernet” network controller) or 0x028000 (indicating “other” network controller), depending upon the value of *otherSubclass* in the **PciParm** entry of the EEPROM.

LatencyTimer

This 8-bit read/write register specifies, in units of PCI bus clocks, the value of the latency timer for bus master operations.

The register format is as follows:



The system writes a value into **LatencyTimer**, which determines how long the adapter may hold the bus in the presence of other bus requesters. Whenever the adapter asserts FRAME#, the latency timer is started. When the timer count expires, the adapter must relinquish the bus as soon as its GNT# signal has been negated.

Since the low-order three bits are not implemented, the granularity of the timer is eight bus clocks.

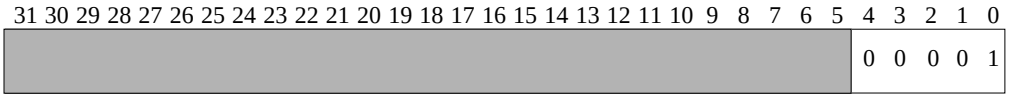
HeaderType

The value returned in this read-only field, 0x00, identifies the adapter as a single-function PCI device and specifies the configuration register layout shown in Figure 5-1.

IoBaseAddress

This read/write register allows the system to define the I/O base address for the adapter. PCI requires that I/O base addresses be set as if the system used 32-bit I/O addressing. The register returns 1 in bit [0] to indicate that this is an I/O base address. The upper 27 bits of the register can be written to, indicating that the adapter requires 32 bytes of I/O space in the system I/O map.

The register format is as follows:

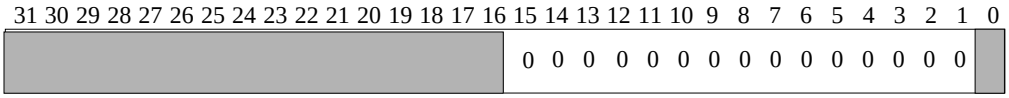


ioBaseAddress [31:5]: The system programs the I/O base address into this field. Since the adapter uses 32 bytes of I/O space, 27 bits are required to completely specify the I/O base address.

BiosRomControl

This read/write register allows the system to define the base address for the adapter's BIOS ROM.

The register format is as follows:



romBaseAddress [31:16]: The system programs the expansion ROM base address into this field.

Since this field is 16 bits wide, the ROM can be mapped on 64 KB boundaries. If a ROM smaller than 64 KB is installed, it appears as multiple images within the 64 KB space.

addressDecodeEnable [0]: When this bit is cleared, the adapter's BIOS ROM is disabled. Setting this bit causes the adapter to respond to accesses in its configured expansion ROM space, if *memorySpace* in the **PciCommand** register is also set.

InterruptLine

This 8-bit read/write register is written by the system to communicate to the device driver which interrupt level is being used for the device. This allows the driver to use the appropriate interrupt vector for its ISR.

For 80x86 systems, the value in **InterruptLine** corresponds to the IRQ numbers (0 through 15) of the standard dual 8259 configuration, and the value 255 corresponds to "disabled."

InterruptPin

This read-only register indicates which PCI interrupt "pin" the adapter will use. PCI bus master adapters will always use INTA#, so 0x01 is returned in **InterruptPin**.

MinGnt

This read-only value specifies in 250 ns increments how long a burst period the adapter requires when operating as a bus master. This value is used as a clue to the system in setting the **LatencyTimer**.

The PCI bus master adapter returns the value 3 in this field. This assumes a 33 MHz bus (30 ns clock period), one clock for the address phase, a three clock latency to the first data phase, and then no wait states for the 15 remaining data phases.

[1 + 3 + 16] 30 ns = 600 ns (round up to 750 ns)

MaxLat

This read-only value specifies in 250 ns increments how often the adapter requires the bus when operating as a bus master. This value is used as a clue to the system in setting the **LatencyTimer.**

The bus master PCI returns the value 8 in this field, implying a latency tolerance of 2 µs. In order to keep up with an incoming frame, the adapter needs to transfer 64 bytes (assuming a 16 dword burst length) every

64 bytes / 12.5 MBps = 5.12 µs

This implies using a **MaxLat** value of about 20. To do much better than just keep up with the incoming frame, lower the value to 8.

ResetOptions

ResetOptions is a read-only register that shows the media options available on the adapter. This register is also available as a read/write register in Window 3 of the I/O space.

Refer to the section “ResetOptions” on page 4-38 for more information about **ResetOptions**.

InternalConfig

This read/write register allows for reading and setting adapter options other than those set in the predefined PCI configuration registers. The register format is identical to the **InternalConfig** register in Window 3 of the I/O space.

InternalConfig is loaded with a default value from EEPROM location 0x12 or 0x15 after system reset. Driver/configuration software can then write a new value in **InternalConfig** and save the new value to EEPROM, if desired.

For more information about **InternalConfig**, refer to the section “InternalConfig” on page 4-21.

EISA Configuration Overview

EISA systems provide 1 KB of slot-specific I/O space per slot. Within this space, EISA adapters are expected to supply configuration registers starting at address `0zC80`, where `z` is the slot number.

3Com EISA bus master adapters unconditionally decode accesses to the EISA configuration register range, `0zC80` to `0zCFF`, and map those accesses into the Window 0 through Window 7 I/O registers according to the following table.

Address	Window
0zC80-F	0
0zC90-F	1
0zCA0-F	2
0zCB0-F	3
0zCC0-F	4
0zCD0-F	5
0zCE0-F	6
0zCF0-F	7

Normally, the EISA system BIOS checks the **ManufacturerId/ProductId** against its current configuration in nonvolatile RAM. If the adapter is newly installed, the user is prompted to run the EISA configuration program, supplied by the PC manufacturer. EISA configuration files (which have a filename extension ".CFG") are supplied with EISA adapters to allow options such as interrupt level and BIOS PROM to be configured without conflict.

After reading **ManufacturerId/ProductId**, the system BIOS configures EISA bus master adapters by writing values into the **AddressConfig** and **ResourceConfig** registers. These settings are also saved in the system's nonvolatile RAM for subsequent boot sequences.

The adapter is then enabled by setting the *cardEnable* bit in the **ConfigControl** register. This causes the adapter to map the switchable I/O register window at slot-specific addresses 0z000 through 0z00F, and to map the fixed Window 1 decode at 0z010 through 0z01F.

EISA Configuration Registers

This section contains brief descriptions of the EISA configuration registers. Unlike PCI, the EISA configuration registers are part of the regular I/O register set, and are visible through the standard I/O register window mechanism. For complete information on these registers, refer to the definitions in Chapter 4, “Register Listings.”

ManufacturerId Contains the read-only EISA manufacturer’s code that is assigned to 3Com.

ProductId Contains the read-only product code that identifies individual adapter products.

ConfigControl Contains the writable *cardEnable* bit, which the EISA system sets to allow the adapter to decode addresses in the slot-specific address range `0z000` through `0z01F`.

AddressConfig Contains a field to allow the system to set a base address for the on-board BIOS ROM.

ResourceConfig Contains a field to allow the system to assign an interrupt level to the adapter.

EEPROM Data Format

This section contains reference information on the EEPROM contents.

PCI Data Format

Table 5-1 summarizes the contents of the EEPROM.

Table 5-1. PCI EEPROM Data Format

Offset (hex)	Field Name	10/100 Default (hex)	10 Only Default (hex)
00	3Com Node Address (word 0)	0020	0020
01	3Com Node Address (word 1)	afxx	afxx
02	3Com Node Address (word 2)	xxxx	xxxx
03	DeviceId	5950	5900
04	Manufacturing Data - Date	xxxx	xxxx
05	Manufacturing Data - Division	00xx	00xx
06	Manufacturing Data - Product Code	xxxx	xxxx
07	Manufacturer Id	6d50	6d50
08	PciParm	0418	0418
09	RomInfo	0000	0000
0a	OEM Node Address (word 0)	0020	0020
0b	OEM Node Address (word 1)	afxx	afxx
0c	OEM Node Address (word 2)	xxxx	xxxx

(continued)

Table 5-1. PCI EEPROM Data Format (continued)

Offset (hex)	Field Name	10/100 Default (hex)	10 Only Default (hex)
0d	Software Information	3f10	3f10
0e	Compatibility Word	0000	0000
0f	Software Information2	0000	2000
10	CapabilitiesWord	11c6	01c6
11	Reserved	0000	0000
12	InternalConfig Low	001b	0012
13	InternalConfig High	0101	0102
14	Reserved	0000	0000
15–16	Reserved	0000	0000
17	Checksum	00yy	00yy

EISA Data Format

Table 5-2 summarizes the contents of the EEPROM.

Table 5-2. EISA EEPROM Data Format

Offset (hex)	Field Name	10/100 Mbps Default (hex)	10 Mbps Only Default (hex)
00	3Com Node Address (word 0)	0020	0020
01	3Com Node Address (word 1)	afxx	afxx
02	3Com Node Address (word 2)	xxxx	xxxx
03	ProductId	7059	2059
04	Manufacturing Data - Date	xxxx	xxxx
05	Manufacturing Data - Division	00xx	00xx
06	Manufacturing Data - Product Code	xxxx	xxxx
07	ManufacturerId	6d50	6d50
08	AddressConfig	0000	0000
09	ResourceConfig	3000	3000
0a	OEM Node Address (word 0)	0020	0020
0b	OEM Node Address (word 1)	afxx	afxx
0c	OEM Node Address (word 2)	xxxx	xxxx
0d	Software Information	3f10	3f10
0e	Compatibility Word	0000	0000
0f	Software Information2	0000	0000
10	CapabilitiesWord	11c6	01c6
11	Reserved	0000	0000
12	InternalConfig Word 0	001b	0012
13	InternalConfig Word 1	0101	0102
14–16	Reserved	0000	0000
17	Checksum	00xx	00xx

Data Field Details

3Com Node Address

This field contains the 3Com node address for the adapter. This is **not** the field to be programmed into the **StationAddress** register. Refer to the section "OEM Node Address" on page 5-11.

DeviceId (PCI Only)

This is the two-byte product identifier, which gets loaded into the ASIC and made available in the **DeviceId** register in the PCI configuration space.

The most significant three nibbles are the numeric portion of the 3Com "3C" number. The least significant nibble is used as a revision code to reflect the particular transceiver resources on the adapter and potential ASIC revisions. The numbers have been defined for the adapters indicated, as shown in Table 5-3.

Table 5-3. Code Numbers for 3Com 3C Numbers

Code	Adapter Type	Connector Type
5950	Fast EtherLink PCI	Shared 10BASE-T/100BASE-TX
5951	Fast EtherLink PCI	Shared 10BASE-T/100BASE-T4
5952	Fast EtherLink PCI	10BASE-T/MII
5900	EtherLink III PCI	10 Mbps (10BASE-T, 10BASE2, AUI) 10BASE-T only

ProductId (EISA Only)

This is the two-byte product identifier required by the EISA bus. It is read automatically by the adapter upon reset, and made available in the **ProductId** register.

Manufacturing Data - Date

This is the date of manufacture, encoded according to the following:

day [4:0]: The day (1 through 31).

month [8:5]: The number of the month (1 through 12).

year [15:9]: The last two digits of the current year (0 through 99).

Manufacturing Data - Division

This is the manufacturing division code, as shown on the product bar code label.

Manufacturing Data - Product Code

This is the manufacturing product code, which is two alphanumeric ASCII characters from the bar code label.

ManufacturerId

This is 3Com's assigned EISA manufacturer ID. It is a byte-swapped, encoded form of the string "TCM."

***NOTE**: For PCI adapters, this value has no significance in PCI operation (it is unrelated to the PCI* **VendorId** *value). It is not used by the adapter logic in any way, nor is it made available in any adapter I/O register.*

PciParm (PCI Only)

This is loaded into the ASIC and controls various hardware functions related to PCI bus operation.

fastBackToBack	[0]: Determines the value for the *fastBackToBack* bit in the **PciStatus** register.
udfSupported	[1]: Determines the value for the *udfSupported* bit in the **PciStatus** register.
otherSubclass	[2]: Determines which subclass code is returned in the **ClassCode** register. 0 is the Ethernet subclass; 1 is the Other subclass.
minGnt	[6:3]: Determines the value returned in the low-order bits of the **MinGnt** register.
maxLat	[10:7]: Determines the value returned in the low-order bits of the **MaxLat** register.
unassigned	[15:11] Unassigned and should be written with zeroes.

RomInfo (PCI Only)

This informs a driver or configuration program whether a BIOS ROM is installed, and the physical size of the ROM.

romSize	[13:12]: The **physical** size of the BIOS ROM. The *romSize* bit is valid only when *romPresent* is set.
romPresent	[11]: Indicates the presence of a BIOS ROM.

OEM Node Address

This is the field to be programmed into the **StationAddress** register. For 3Com adapters, this field will contain the same value as in 3Com Node Address. OEM customers may choose to program this field with a different value.

AddressConfig (EISA Only)

This field supplies the value for the **AddressConfig** register. It is read automatically by the hardware upon reset to provide default settings for ROM settings and the *autoSelect* bit. See the **AddressConfig** register definition for bit definitions.

ResourceConfig (EISA Only)

This field supplies the value for the **ResourceConfig** register. It is read automatically by the hardware upon reset to provide a default setting for the interrupt level. See the **ResourceConfig** register definition for bit definitions.

Software Information

This field contains environmental information for use by the driver. The default value specifies optimize for DOS client, 500 µs disable time, link beat enabled.

reserved	[3:0]: Reserved, set to zero.
optimizeFor	[5:4]: Specifies the environment for which to optimize. 00: Reserved 01: DOS clients 10: Multitasking clients 11: Servers
reserved	[7:6]: Reserved, set to zero.
intDisableTime	[13:8]: Specifies the maximum time the host can disable interrupts, according to the formula: max interrupt disable time = (25 + *intDisableTime* * 25) µs
linkBeatDisable	[14]: Indicates to the host software whether it should set *linkBeatEnable* in **MediaStatus** (for appropriately equipped adapters). Note the opposite polarities of these bits.
reserved	[15]: Reserved, set to zero.

Compatibility Word

This field contains two byte-wide values that are checked by the driver with an internal value (CLevel) to determine the compatibility of the driver with the software.

warningLevel	[7:0]: If the driver's CLevel is less than this field, the driver issues a warning message that a newer driver is available that may offer improved performance.
failureLevel	[15:8]: If the driver's CLevel is less than this field, the driver fails the install process. A new driver needs to be obtained.

Capabilities Word

This word contains data defining the basic capabilities of the adapters. The following table summarizes the capabilities of the PCI/EISA adapter. The resulting value is 0x11C6.

Bit	Capabilities Bit	Value
0	*supportsPlugNPlay*	0
1	*supportsFullDuplex*	1
2	*supportsLargePackets*	1
3	*supportsSlaveDma*	0
4	*supportsSecondDma*	0
5	*supportsFullBusMaster*	0
6	*supportsFragBusMaster*	1
7	*supportsCrcPassThru*	1
8	*supportsTxDone*	1
9	*supportsNoTxLength*	0
10	*supportsRxRepeat*	0
11	*supportsSnooping*	0
12	*supports100Mbps*	*
13	*supportsPowerMgmt*	0

* This value is 0 for 10 Mbps only or 1 for 10/100 Mbps.

The following paragraphs describe the bits that are set for the adapter.

supportsFullDuplex [1]: Indicates that the adapter supports full-duplex media operation.

supportsLargePackets [2]: Indicates whether the adapter supports frame sizes over 2,047 bytes.

supportsFragBusMaster [6]: Indicates whether the adapter supports fragment bus master data transfers.

supportsCrcPassThru [7]: Indicates whether the adapter supports CRC passthrough using the *crcAppendDisable* bit in the frame start header.

supportsTxDone [8]: Indicates whether the adapter supports the `TxDone` command.

supports100Mbps [12]: Indicates the adapter's ability to support 100 Mbps data rates.

InternalConfig

These two words supply the value for the **InternalConfig** register. The low word is bits [15:0]; the high word is [31:16] bits. They are read automatically by the hardware upon reset to provide default settings for non-system-related configuration settings. The value can later be written over by driver software.

Refer to the **InternalConfig** register definition for bit definitions.

Software Information 2

This field contains additional information for drivers as follows:

reserved	[4:0]: Reserved, set to zero.
noRxOvnAnomaly	[5]: 10 Mbps receive overrun anomaly. Set to 1, the adapter does not contain the 10 Mbps receive overrun anomaly; set to 0, the adapter contains the 10 Mbps anomaly. (PCI only; for EISA, this bit is undefined and will always be zero.)
reserved	[15:6]: Reserved, set to zero.

Checksum

The checksum is a byte-wise XOR computed across component bytes in words 0 through 16.

Appendix A
Errata List and Software Solutions

Introduction

Over time anomalies in the PCI/EISA bus-master family of adapters have been found. Software solutions for these are documented below. Fixes for these anomalies have been incorporated into the hardware over various revisions of these adapters. Therefore, to ensure that your driver will run on all revisions, you must incorporate all the workarounds listed. 3Com cannot guarantee compatibility of any driver that does not contain all of these software solutions.

The following software workarounds are required or highly recommended for the PCI/EISA bus master family of adapters. The initialization workarounds are unique to the bus. The run-time workarounds are common to both families, except as noted below.

PCI Adapters (3C590 and 3C595):

1 - The adapter's response to a parity error must be disabled. This is done by clearing the parityErrorResponse bit in the PCI PciCommand configuration register (this bit is also cleared upon system reset, but it is usually set by the BIOS).

2 - Bus Master capability must be enabled prior to using bus master operation. This is done by setting the busMaster bit in the PCI PciCommand configuration register.

3 - The PCI LatencyTimer configuration register must be set to the maximum value (0xFF). This is to avoid a data corruption issue if the timer elapses while a bus master operation is in progress.

4 - There is a bug in the 10 Mbit receive logic of the first revision of the adapter which causes overruns. The workaround is complex and difficult to implement. It is currently available as a separate document.

5 - Due to a PCI signaling issue, the adapter can experience data corruption in some machines when a bus master transfer crosses a 4K memory address boundary. To avoid this, the driver should perform PIO transfers in the 4-byte region at the end of a 4K boundary.

Errata 1, 3, and 4 above have been fixed in newer version of the adapter. The adapters with this fix can be identified by examining bit 5 in the Software Information 2 field of the EEPROM. If this bit is one, the errata do not apply. If this bit is zero, the associated workarounds must be implemented.

EISA Adapters (3C592 and 3C597):

1 - Bus Master operations must start on 32-byte boundaries (physical memory address). If the buffer's memory address is not on a 32-byte boundary, use Programmed I/O to transfer the data until such a boundary is reached.

2 - A bus master reset is required prior to starting a bus master operation. Clear the dmaReset bit (while masking off other Resets) in the GlobalReset command, and remember to spin on commandInProgress in the IntStatus register to allow the reset to complete before doing other programming operations to the adapter.

3 - Some computers violate the EISA bus timing specification relating to when Bus Master Address and Length registers are written. The workaround is to read back the data that's written, compare the data read to the written data, and write the data again when a miscompare occurs. This happens rarely, so the compare loop will not loop indefinitely.

4 – An IO write operation immediately followed by a statistics register read may hang the adapter. The workaround is to perform an IO read (from any register, for example, Command/ Status) before doing a Statistics Read. This ensures that n write has preceded this stats read.

5 - When a receive bus master operation is programmed with RxEarly enabled and there is no data to transfer, receive pacing logic in the adapter causes it to wait until data has been received from the network. A problem occurs in this logic under special circumstances. The problem happens during a bus master upload of a received packet. The adapter incorrectly tags the last data burst of the frame as having filled the 64-byte internal cache instead of being partially filled. The result is too much data is transferred during the upload (up to 63 bytes), and MasterLen is up to 63 bytes too small. This may result in memory corruption. There are three ways to work around this problem:

1) Disable RxEarly interrupts and only accept Receive Packet Complete interrupts. This disables the pacing logic.

2) If the size of the packet is known, program the bus master length register to upload the correct number of bytes.

3) Don't allow bus master uploads to cross a 1K (400 hex) physical address (stopping/re-starting on a 1K boundary is OK).

This problem is fixed in the second revision of the EISA ASIC. The fixed adapters can be detected by examining bit 6 of Software Information 2 field in the EEPROM. If bit 6 is set to zero, the problem exists. If bit 6 is set to one, this adapter doesn't have the problem. So, an alternative would be to disallow use of the pre-rev2 boards and notify the user that they must obtain a later revision of the adapter.

All Adapters

1 - When the statistics interrupt occurs, the badSSD count must be read from the diag window as part of the statistics read.

2 - Bus master receive data uploads may "hang" the adapter under certain circumstances. This bug appears to occur only at 10 Mbps. When the hang is detected, all of the data is seen to have been transferred correctly to system memory, but the bus master logic is stuck in one of two states. A host reset (GlobalReset command with hostReset unmasked) will clear the hang condition.

Workaround: avoid "pacing" on uploads (start an upload DMA transfer only after rxComplete is set) or always make sure the driver guesses the packet size correctly, so that uploads complete due to MasterLen countdown rather than end-of-packet.

3 - Drivers should set RxEarlyThresh to 60 bytes or greater to avoid a problem in which an rxComplete interrupt is generated, but when the driver checks the adapter, there is no receive packet available because the receive FIFO is empty. (The adapter discards packets less than 60 bytes long, such as from collisions.)

4 - When performing PIO reads, drivers should not read the receive FIFO within a DWORD (4 bytes) of the end of the data unless unless rxComplete is true. In other words, until rxComplete, if rxBytes returns, for instance, 16 bytes, the driver might only read 8, leaving data in FIFO.

Useful Tips

Note that the parameter for the SetTxAvailThresh command is shifted up 2 bits before it is placed into the TxStartThreshold register (Window 5, offset 0), effectively multiplying the parameter by four. A drive should terefore divide the desired threshold value by four when generating the parameter value for this command.

Useful Tips