TEXAS INSTRUMENTS

ThunderLAN

TNETE100A, TNETE110A, TNETE211

Programmer's Guide

October 1996

Network Business Products

Texas Instruments

Printed in U.S.A., October 1996
L411001–9761 revision A

SPWU013A

Programmer's Guide

ThunderLAN

TNETE100A , TNETE110A , TNETE211

1996

ThunderLAN™ Programmer's Guide

TNETE100A, TNETE110A, TNETE211

Literature Number: SPWU013A
Manufacturing Part Number: L411001-9761 revision A
October 1996

TEXAS INSTRUMENTS

IMPORTANT NOTICE

Texas Instruments (TI) reserves the right to make changes to its products or to discontinue any semiconductor product or service without notice, and advises its customers to obtain the latest version of relevant information to verify, before placing orders, that the information being relied on is current.

TI warrants performance of its semiconductor products and related software to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

Certain applications using semiconductor products may involve potential risks of death, personal injury, or severe property or environmental damage ("Critical Applications").

TI SEMICONDUCTOR PRODUCTS ARE NOT DESIGNED, INTENDED, AUTHORIZED, OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT APPLICATIONS, DEVICES OR SYSTEMS OR OTHER CRITICAL APPLICATIONS.

Inclusion of TI products in such applications is understood to be fully at the risk of the customer. Use of TI products in such applications requires the written approval of an appropriate TI officer. Questions concerning potential risk applications should be directed to TI through a local SC sales office.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards should be provided by the customer to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance, customer product design, software performance, or infringement of patents or services described herein. Nor does TI warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used.

Preface

Read This First

About This Manual

The *ThunderLAN Programmer's Guide* assists you in using the following implementations of ThunderLAN networking hardware:

- TNETE100A Ethernet™ controller
- TNETE110A Ethernet controller
- TNETE211 100 VG-AnyLAN physical media interface (PMI)

How to Use This Manual

The goal of this book is to assist you in the development of drivers for the ThunderLAN controllers. This document contains the following chapters:

- Chapter 1, ThunderLAN Overview, describes some Texas Instruments-specific hardware features. These include the enhanced media independent interface (MII), which passes interrupts from an attached physical interface (PHY) to the host.
- Chapter 2, ThunderLAN Registers, shows how to access the various ThunderLAN registers and how to use these registers to access external devices attached to ThunderLAN.
- Chapter 3, Initializing and Resetting, discusses how to initialize and reset the controller and the attached PHYs.
- Chapter 4, Interrupt Handling, describes what happens when interrupts occur and how to correct failure conditions.
- Chapter 5, List Structures, describes how to pass data to ThunderLAN using a system of linked list structures.
- Chapter 6, Transmitting and Receiving Frames, explains the format and procedure for transmitting and receiving, as well as the linked list structure required.
- Chapter 7, Physical Interface, discusses the features of ThunderLAN which support IEEE 802.3- and 802.12-compliant devices.

Notational Conventions

This document uses the following conventions:

- Program listings, program examples, and interactive displays are shown in a special font. Examples use a bold version of the special font for emphasis. Here is a sample program listing:

```
11 0005   0001          .field    1,     2
12 0005   0003          .field    3,     4
13 0005   0006          .field    6,     3
14 0006                 .even
```

- A lower case ‘x’ in a number indicates that position can be anything (don’t care). Here are some examples:
  - 0x00
  - 0x0004
  - 0x4000501

Related Documentation

Information Technology Local and Metropolitan Area Networks – Part 12: Demand-Priority Access Method, Physical Layer and Repeater Specifications for 100-Mb/s Operation, Draft 8.0 of the Revision Marked for Technical changes of IEEE Standard 802.12.

MAC Parameters, Physical Layer, Medium Attachment Units and Repeater for 100-Mb/s Operation, Draft 5.0 of the Supplement to 1993 version of ANSI/IEEE Std. 802.3: Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method & Physical Layer Specifications.

PCI Local Bus Specification*, *Revision 2.0 is the specification which ThunderLAN is designed to meet. To obtain copies, contact PCI Special Interest Group, P.O. Box 14070, Portland, OR 97214, 1–800–433–5177.

ThunderLAN Adaptive Performance Optimization Technical Brief (Texas Instruments literature number SPWT089) discusses specific buffering and pacing techniques for improving adapter performance by adjusting the resources and transmit procedures to achieve optimal transmission rate and minimal CPU use.

XL24C02 Data Sheet, EXEL Microelectronics, 1993, which contains the device specifications for the XL24C02 2M-bit electrically erasable EPROM.

If You Need Assistance. . .

❑ World-Wide Web Sites

TI Online	http://www.ti.com
Semiconductor PIC	http://www.ti.com/sc/docs/pic/home.htm
Networking Home Page	http://www.ti.com/sc/docs/network/nbuhomex.htm

❑ North America, South America, Central America

Product Information Center (PIC)	(972) 644-5580	
TI Literature Response Center U.S.A.	(800) 477-8924	
Software Registration/Upgrades	(214) 638-0333	Fax: (214) 638-7742
U.S.A. Factory Repair/Hardware Upgrades	(713) 274-2285	
U.S. Technical Training Organization	(972) 644-5580	
Networking Hotline		Fax: (713) 274-4027 Email:TLANHOT@micro.ti.com

❑ Europe, Middle East, Africa

European Product Information Center (EPIC) Hotlines:

Multi-Language Support	+33 1 30 70 11 69	Fax: +33 1 30 70 10 32 Email: epic@ti.com
Deutsch +49 8161 80 33 11	or +33 1 30 70 11 68	
English	+33 1 30 70 11 65	
Francais	+33 1 30 70 11 64	
Italiano	+33 1 30 70 11 67	
EPIC Modem BBS	+33 1 30 70 11 99	
European Factory Repair	+33 1 93 22 25 40	
Europe Customer Training Helpline		Fax: +49 81 61 80 40 10

❑ Asia-Pacific

Literature Response Center	+852 2 956 7288	Fax: +852 2 956 2200

❑ Japan

Product Information Center	+0120-81-0026 (in Japan)	Fax: +0120-81-0036 (in Japan)
	+03-3457-0972 or (INTL) 813-3457-0972	Fax: +03-3457-1259 or (INTL) 813-3457-1259

❑ Documentation

When making suggestions or reporting errors in documentation, please include the following information that is on the title page: the full title of the book, the publication date, and the literature number.

Mail: Texas Instruments Incorporated
Technical Documentation Services, MS 702
P.O. Box 1443
Houston, Texas 77251-1443

Email: comments@books.sc.ti.com

Note: When ordering documentation from a Literature Response Center, please specify the literature number of the book.

Trademarks

Ethernet is a trademark of Xerox Corporation.

ThunderLAN and Adaptive Performance Optimization are trademarks of Texas Instruments Incorporated.

Contents

Figures

Tables

Chapter 1

ThunderLAN Overview

The ThunderLAN family consists of highly integrated, single-chip networking hardware. It uses a high-speed architecture that provides a complete peripheral component interconnect (PCI)- to-10Base-T/AUI (adapter unit interface) Ethernet solution. It allows the flexibility to handle 100M-bps Ethernet protocols as the user's networking requirements change.

The TNETE100A, one implementation of the ThunderLAN architecture, is an intelligent protocol network interface. Modular support for the 100 Base-T (IEEE 802.3u) and 100VG-AnyLAN (IEEE 802.12) is provided via a media independent interface (MII). The TNETE110A is the same device without the MII and is 10M bps only. ThunderLAN uses a single driver suite to support multiple networking protocols.

ThunderLAN architecture was designed to achieve the following goals:

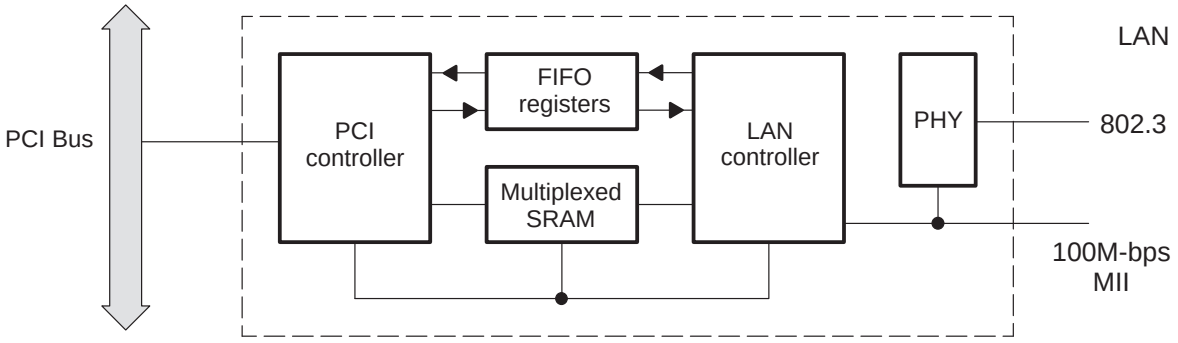
- ❏ High performance with low use of host CPU
- ❏ Simplicity of design
- ❏ Ease of upgrade to higher speed networks
- ❏ Freedom of choice of network protocol

ThunderLAN allows a simple system design by integrating a PCI controller, an internal first in, first out (FIFO) buffer, a LAN controller, and a 10Base-T physical interface (PHY).

Topic **Page**

1.1 ThunderLAN Architecture

Figure 1–1. The ThunderLAN Controller



An integrated PHY provides interface functions for 10Base-T carrier sense multiple access/collision detect (CSMA/CD) (Ethernet). A MII is used to communicate with the integrated PHY. The PHY is an independent module from the rest of the ThunderLAN controller. This allows the PHY to be reset and placed in a power-down mode.

The PCI controller is responsible for direct memory accesses (DMAs) to and from the host memory. It is designed to relieve the host from time-consuming data movements, thereby reducing use of the host CPU. The PCI interface supports a 32-bit data path.

ThunderLAN supports two transmit and one receive channels. The demand priority protocol supports two frame priorities: normal and priority. The two transmit channels provide independent host channels for these two priority types. CSMA/CD protocols only support a single frame priority, but the two channels can be used to prioritize network access, if needed. All received frames pass through the single receive channel.

ThunderLAN's multiplexed SRAM is 3.375K bytes in size. This allows it to support one 1.5K byte FIFO for receive, two 0.75K byte FIFOs for the two transmit (Tx) channels, and three 128-byte lists (see section 5.1, List Management). In one-channel mode, the two Tx channels are combined, giving a single 1.5K-byte FIFO for a single Tx channel. Supporting 1.5K byte of FIFO per channel allows full frame buffering of Ethernet frames. PCI latency is such that a minimum of 500 bytes of storage is required to support 100M-bps LANs. (Refer to the *PCI Local Bus Specification,* revision 2.0, section 3.5, Latency).

ThunderLAN's industry-standard MII permits ease of upgrade. External devices can be connected to the MII and managed, if they support the two-wire management interface. PHY layer functions for 100M-bps CSMA/CD and demand priority are connected to the MII.

1.2 Networking Protocols

The MII also allows freedom in choosing a networking protocol. It allows the use of standard 100M bps CSMA/CD PHY chips. ThunderLAN uses these signal lines to interface to an external 100M bps demand priority PHY. This gives ThunderLAN the flexibility necessary to handle 10Base-T, 10Base-2, 10Base-5 AUI, 100Base-TX, 100Base-T4, 100Base-FX, and 100VG-AnyLAN today, while supporting emerging technologies.

ThunderLAN is designed to simplify the software used to transmit frames, receive frames, and service the PHY events. It accomplishes this by integrating time-consuming tasks into the controller. These tasks include:

- The DMA of data into and out of the controller
- A simplified, interrupt-driven frame buffer management technique
- The elimination of PHY register polling through MII interrupts

DMA of data is handled through list structures. ThunderLAN's method of handling data through list structures has parallels with the method used in Texas Instruments TI380 COMMprocessors. There are some differences, such as the use of a 0 forward pointer.

ThunderLAN is designed to meet *PCI Local Bus Specification*, revision 2.0 for its PCI interface standards.

1.3 PCI Interface

The PCI local bus is a high-performance, 32- or 64-bit bus with multiplexed address and data lines. The bus is designed to be a medium between highly integrated peripheral controller components such as ThunderLAN, add-in boards, and processor/memory systems.

1.3.1 PCI Cycles

ThunderLAN executes the following cycles when it acts as the PCI bus master. The hexadecimal number shown is the bus command encoded in the PC/BE[3::0]# signals.

- 0x7h–memory write
- 0xCh–memory read multiple
- 0xEh–memory read line

ThunderLAN responds to the following PCI cycles when acting in slave mode on the PCI bus:

- 0x2h–I/O read
- 0x3h–I/O write
- 0x6h–memory read
- 0x7h–memory write
- 0xAh–configuration read
- 0xBh–configuration write
- 0xCh–memory read multiple
- 0xEh–memory read line
- 0xFh–memory write and invalidate

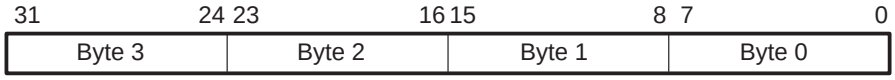
Future versions of ThunderLAN may not be limited to these PCI cycles. Texas Instruments reserves the right to add or delete any cycles to the ThunderLAN PCI controller. When designing a system, ensure that the attached interface to ThunderLAN is fully compliant with the *PCI Local Bus Specification*.

1.3.2 Byte Ordering

ThunderLAN follows the *PCI Local Bus Specification* convention when transferring data on the PCI bus. The PCI bus data is transferred on the PAD[31::0] lines. PAD31 is the most significant bit, and PAD0 is the least significant bit.

The 32 data lines are enough to transfer four bytes per data cycle. Byte 0 is the LSbyte and byte 3 is the MSbyte. Byte 0 uses bits 0–7, byte 1 uses bits 8–15, byte 2 uses bits 16–23, byte 3 uses bits 24–31.

Figure 1–2. PCI Bus Byte Assignment



ThunderLAN uses the full four bytes per data cycle. The only exception is when the data to be transferred is not octet aligned. In this case, the PCI controller might not transfer the full four bytes on the first cycle. ThunderLAN deasserts the IRDY signal only once, if needed, to synchronize the PCI bus to the internal 64-bit architecture. The deassertion of IRDY occurs on the third cycle of the PCI bus. ThunderLAN does not deassert IRDY for the rest of the transfer unless the PCI bus asserts the TRDY signal.

Chapter 2

ThunderLAN Registers

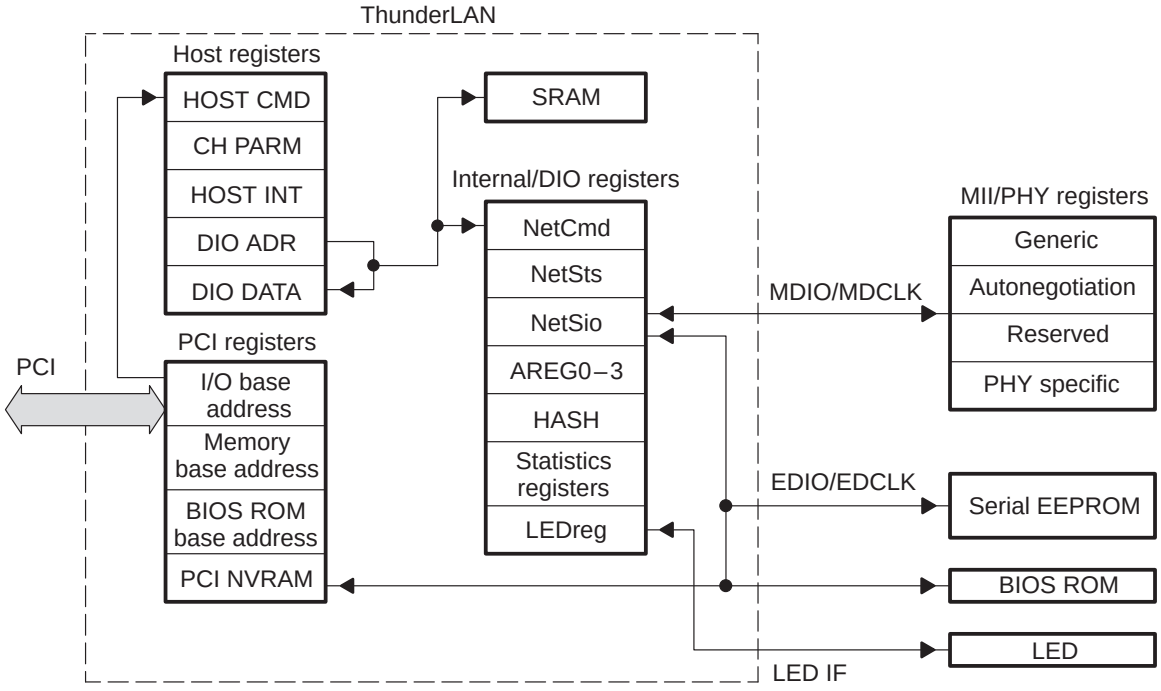
ThunderLAN uses a variety of registers to perform its networking functions. These include peripheral component interface (PCI) registers, host registers, internal direct input/output (DIO) registers, media independent interface (MII) registers, and physical interface (PHY) registers. Access to these is a requirement for setting up the ThunderLAN controller and any of the PHY devices attached to the MII. They must be accessed as well for transmission, initiation, and reception of data. Other activities which require the user to understand ThunderLAN's register spaces include determining the cause of event-driven interrupts and how to clear them and diagnostic functions. This chapter explains register configurations and discusses control of these spaces through code examples.

Topic **Page**

2.1 Register Addresses

The following figure shows the various register spaces provided by ThunderLAN. It also shows how a driver uses ThunderLAN's registers to interface to external devices such as PHYs, BIOS ROMs, and EEPROMs.

Figure 2–1. How ThunderLAN Registers are Addressed



One block of registers, the host registers, appear at a programmable place in memory or port address space, directly on the PCI bus. The beginning address is determined by the value written into the PCI configuration space base address registers. Once the base register's address is determined, ThunderLAN reads and writes to these registers like ordinary memory or I/O ports. Since the ThunderLAN devices are directly connected to the PCI, there is no external decode logic that generates a chip select—all the decode is done internally.

ThunderLAN's internal/DIO registers are accessed via the DIO_ADR and DIO_DATA registers in the host register group. An address is placed in the host DIO_ADR register, and the data to be read or written to the DIO register is read or written to the DIO_DATA register. The internal/DIO register space is referenced indirectly via the host registers to minimize the amount of host address space required to support the ThunderLAN controller. External devices and their data are also reached via indirect reference through the host registers

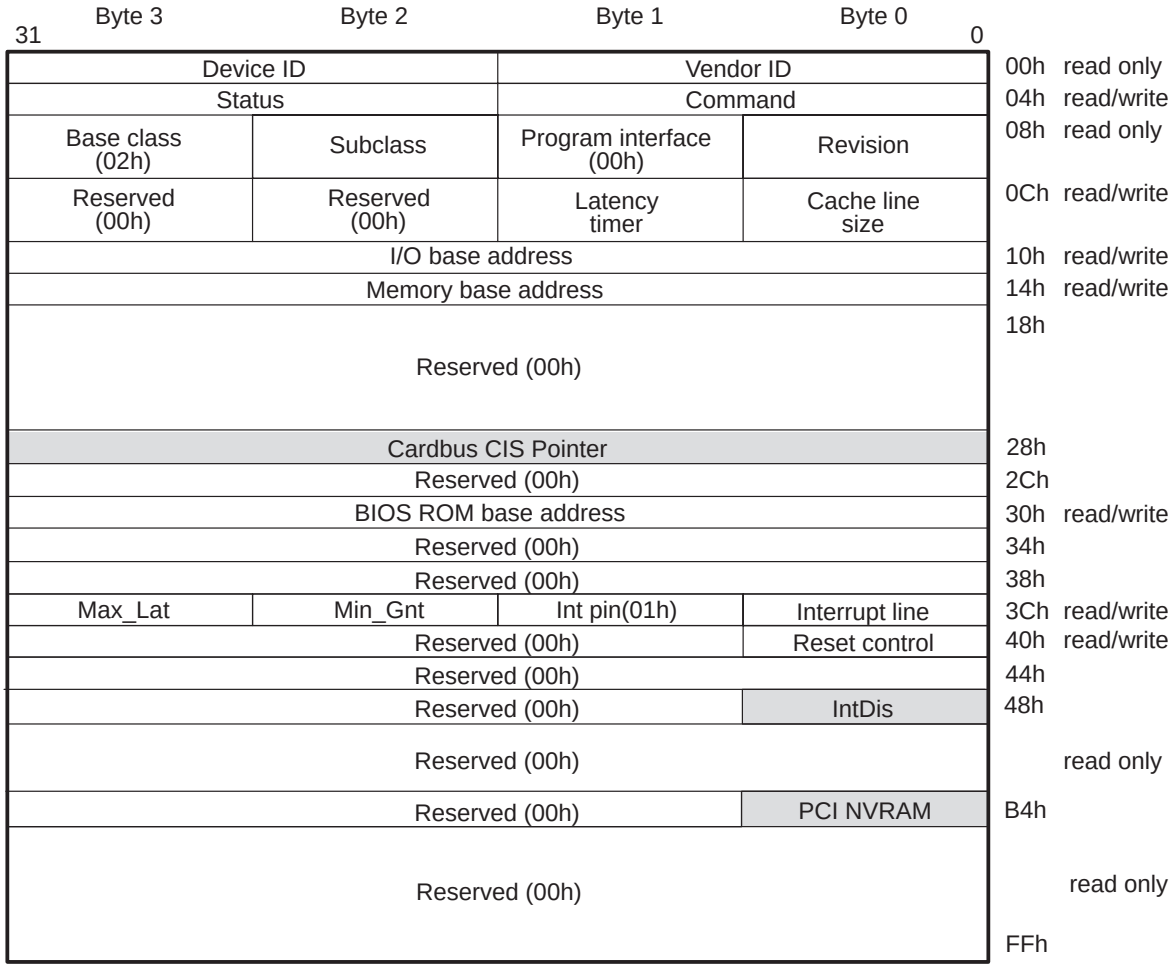
and PCI configuration registers to make control of the system possible through the one PCI interface.

An EEPROM, required by the PCI, can be written to at manufacture time through the PCI_NVRAM register, which is located in the host register space. The EEPROM can also be accessed through the NetSio register which is located in the internal/DIO register space. Control registers on the PHY side of the MII management interface can be similarly written and read through the NetSio register.

A BIOS ROM can be enabled via the BRE bit in the PCI BIOS ROM base address register, and its chip selected address dynamically assigned via a base register in the configuration space. The BRE bit points to a valid address in the ROM address space which causes two byte-address strobe cycles (EALE, EXLE) and a read before the PCI cycle is completed.

2.2 PCI Configuration Space

Figure 2–2. The PCI Configuration Space Registers



Register configuration space information fields are needed to identify a board in a slot to a driver. The functional purpose of the board, the manufacturer, the revision, and several bus requirements can be obtained by inspecting these parameters. The PCI configuration space uses these registers which are called out in the *PCI Local Bus Specification.* These enable the PCI system to:

- ❑ Identify the ThunderLAN controller. This includes setting the interrupt assigned to ThunderLAN.
- ❑ Map the host registers using either the I/O base address register or the memory base address register. The driver uses the address contained in these registers to access ThunderLAN's internal registers.

- Set up the PCI bus. Several PCI bus options can be selected through these registers, including latency and grant. (Refer to *PCI Local Bus Specification,* subsection 3.5)
- Map a BIOS ROM using the BIOS ROM base address register

Many of the registers in the PCI configuration space are accessed with PCI BIOS calls. Refer to the *PCI Local Bus Specification,* chapter 6, for the commands supported by your specific PCI BIOS. Some operating systems (O/Ss) provide BIOS call support. Your operating system's user's guide contains these specific BIOS support routines.

The PCI specification requires that a bus-resident device respond to bus cycle codes reserved for reading and writing to configuration space. See the *PCI Local Bus Specification* document for more information on how these short, slot-dependent address spaces appear to the host processor. The shaded registers in Figure 2–3 can be autoloaded from an external serial EEPROM.

Check the following before accessing the PCI configuration space:

- Ensure that there is a PCI BIOS present or other support for BIOS calls.
- Ensure that the BIOS is the right revision.
- Use a PCI BIOS call to find all attached devices on the PCI bus. Make sure that you are talking to the right device on the PCI bus.

Attaching a pullup resistor to the EDIO pin allows the board designer to automatically read an EEPROM after reset to determine the contents of the first eight bytes, shown shaded below. If the host attempts to read any of the configuration space during the time the adapter is reading the EEPROM, ThunderLAN rejects the request by signaling target-retry.

Figure 2–3. Configuration EEPROM Data Format

	Address
Vendor ID LSByte	C0h
Vendor ID MSByte	C1h
Device ID LSByte	C2h
Device ID MSByte	C3h
Revision	C4h
Subclass	C5h
Min_Gnt	C6h
Max_Lat	C7h
Checksum	C8h

Normally, access to the configuration space is limited to the operating system. On power-up, the vendor ID, device ID, revision, subclass, Min_Gnt, and Max_Lat registers are loaded with default values. Vendor-specific data is loaded into these registers by placing the data into the EEPROM, which is read at the end of reset if autoload is enabled with a pullup resistor on the EDIO pin. If the data read from the EEPROM has a checksum error, values are fetched from the default PCI parameter registers, which are located at addresses 0x08h to 0x0Fh in the internal/DIO registers space.

Some fields in the configuration space like the bits in the memory base address register and the I/O base address register, which indicate the space size allocation required to access the host registers, are hardwired in the ThunderLAN controllers. Some of the allowed PCI configuration space values like base registers beyond the basic I/O and memory base registers are not implemented because no other entities are supported by this PCI interface other than the network function.

To find register information, you must first identify the PCI function ID:

```
//-------------------------------------------------------
//  PCIFindDevice – Find PCI device
//
//  Parameters:
//  DeviceID    WORD The device ID
//  VendorID    WORD The vendor ID
//  Index       WORD index (normally 0, use when more than
1 device)
//   pDev       WORD* Where to put the device id
//
//  Return val:
//   int     0 if successful. see std return codes in
header
//-------------------------------------------------------
WORD PciFindDevice(
    WORD    deviceID,
    WORD    vendorID,
    WORD    Index,
    WORD    *pDev)
{
    union REGS r;
```

```
    r.h.ah = PCI_FUNCTION_ID;
    r.h.al = FIND_PCI_DEVICE;
    r.x.cx = DeviceID;
    r.x.dx = VendorID;
    r.x.si = Index;
    int86(PCI_INT, &r, &r);
    *pDev = (WORD)r.x.bx;
    return (int)r.h.ah;
}
```

This code returns the function ID that is used to request reads and writes to the ThunderLAN PCI configuration space; this varies from installation to installation, based on hardware implementation and slot. This ID is necessary to determine where ThunderLAN is. The device ID indicates a networking card, and the vendor ID is the manufacturer code. These values can be overlaid in the configuration space with values from the EEPROM during the autoconfiguration. These should be available to the driver software either in the BIOS ROM or on machine-readable media supplied with the network board(s).

The following example reads a byte of a PCI register:

```
//-------------------------------------------------------
//  PciRdByte() – Read a byte from PCI configuration space
//
//  Parameters:
//   devid          WORD pci device identifier
//   addr           WORD config address
//
//  Return val:
//   BYTE           value read
//-------------------------------------------------------
BYTE PciRdByte(WORD devid, WORD addr)
{
    union REGS  r;
    r.h.ah = PCI_FUNCTION_ID;   /*  PCI_FUNCTION_ID
0xB1 */
    r.h.al = READ_CONFIG_BYTE;  /*  READ_CONFIG_WORD
0x09 */
    r.x.bx = devid;
```

```
    r.x.di = addr;
    int86(PCI_INT, &r, &r);     /*  PCI_INT        0x1A */
    return (r.x.cx & 0xFF);
}
```

Normally, the constants in this routine (the values assigned to ah, al, and the opcode for the int86 call) are assigned in the header file for the C code. Their values are inserted as comments to enable the reader to resolve the actual values that are used. The device ID, devid, is known to the driver and is used with another PCI O/S call to find the base addresses needed for this call.

```
//--------------------------------------------------------
//  PciRdWord() – PCI read config word
//
//  Parameters:
//   devid          WORD pci device number
//   addr           WORD address to read
//
//  Return val:
//   WORD           value read
//--------------------------------------------------------
WORD PciRdWord(WORD devid, WORD addr)
{
    union REGS r;
    r.h.ah = PCI_FUNCTION_ID;
    r.h.al = READ_CONFIG_WORD;
    r.x.bx = devid;
    r.x.di = addr;
    int86(PCI_INT, &r, &r);
    return(r.x.cx);
}
```

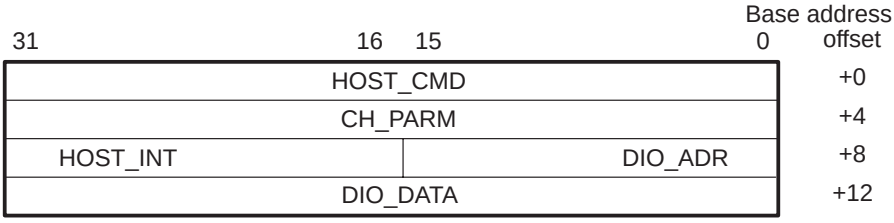
This code passes an address >10 if the driver regards the host registers as memory locations (ThunderLAN's first base address register is hardwired as a memory base register), or >14 if the driver treated the host registers as an I/O block (ThunderLAN's second base register is hardwired as an I/O base register). For the I/O port, the following C instruction could be used to put a value into the DIO_ADR host register:

```
outpw(base_addr+OFF_DIO_ADDR,value);
```

OFF_DIO_ADDR is a constant for the header file. A memory transfer instruction is used if memory space is used instead of I/O space.

2.3 Host Registers

Figure 2–4. Host Registers



ThunderLAN implements the host registers shown above. These are the primary control points for ThunderLAN. Through the host registers, a driver can:

- Reset the ThunderLAN controller
- Start transmit and receive channels
- Handle interrupts: Acknowledge interrupts, turn certain kinds of interrupts on or off, or pace interrupts with the host
- Access the internal registers
- Access the internal SRAM for diagnostic purposes

The HOST_CMD register gives commands to the ThunderLAN controller. It is used in conjunction with the CH_PARM register to start the transmit and receive processes (Tx GO/Rx GO). It is also used in conjunction with the HOST_INT register to acknowledge (ack) interrupts. Through HOST_CMD, interrupt pacing can be selected.

The CH_PARM register is used to give the physical addresses of a transmit or receive list to ThunderLAN's direct memory access (DMA) controller. ThunderLAN uses the address in the CH_PARM register to DMA data into or out of its FIFOs. In an adapter check, an error condition where ThunderLAN must be reset, CH_PARM contains information on the nature of the error.

The HOST_INT register contains information on the type of interrupt that was given to the host processor. It is also used with the CH_PARM register to indicate adapter checks. HOST_INT is designed to make interrupt handling routines simple and powerful. The last two significant bits are set to 0 so that this register may be used as a table offset in a jump table. The bit definitions are mapped to the most significant word (MSW) of the HOST_CMD register. This allows acknowledging of interrupt operations by simply taking the value in HOST_INT and writing it to HOST_CMD.

The DIO_ADR and DIO_DATA registers work in tandem to allow accesses to the internal DIO registers and SRAM. The value in DIO_ADR selects the register or memory locations to be accessed.

To enable reads of adjacent addresses without reposting the address, bit 15 of the DIO_ADR register can be set, which causes the address to be post-incremented by 4 after each access of the DIO_DATA register. This function is useful when reading the statistics or reading the internal SRAM. Autoincrementing while reading the FIFO memory causes a move to the same part of the next 68-bit word; it does not move to the next part of the same 68-bit word. The two least significant bits (LSBs) of the DIO_ADR must be expressly set to get to the various parts of each 68-bit entity.

The host registers are addressed either as memory or I/O ports. The PCI configuration space has locations for the O/S to assign up to six memory or I/O base addresses. The depth of the space requested for each base register implemented is determined by the number of bits, starting at the LSBs, whose values are fixed. The O/S writes to the rest of the bits (with the assumption that the fixed positions are equal to 0) at the beginning address of that block.

As an example, the LSB determines whether the base register is a memory (0) or an I/O space (1) base register. ThunderLAN's PCI interface reserves memory I/O space by implementing an I/O and a memory configuration base register, both with the four LSBs in these registers fixed to indicate the field width requested be reserved in the respective address space for the host register block (four quad words or 16 bytes). The rest of the bits of a base register are filled in by the O/S after all the space requests are considered.

Assigning space in this way assures that all starts of fields are naturally aligned to long words or better. It is important to note that by the time either the BIOS code or driver code is allowed to run, the O/S has queried the card and assigned the base addresses. The host registers can be accessed equally in both address spaces on host processor systems that support both.

Some processors only support memory spaces; in these cases the I/O spaces are assigned a 0, which is not a valid base register value for a peripheral. The driver must check for a 0 base offset value before using the I/O method of accessing the ThunderLAN registers. The base offset must be constant between host processor resets, but can be different for each execution of the program. All host register accesses are done relative to the value found in the respective configuration base register.

The unimplemented base registers in the configuration space return all 0s on a read. This is equivalent to requesting a 2^{32}-byte data space—all of the available address space in a 32-bit address system. PCI interprets an all-bits-fixed situation as not implemented.

2.4 Internal Registers

Figure 2–5. Internal Registers

Byte 3	Byte 2	Byte 1	Byte 0	DIO address
NetMask	NetSts	NetSio	NetCmd	0x00
Man Test		NetConfig		0x04
Default device ID MSbyte	Default device ID LSbyte	Default vendor ID MSbyte	Default vendor ID LSbyte	0x08
Default Max_Lat	Default Min_Lat	Default Subclass	Default revision	0x0C
Areg_0 (23 to 16)	Areg_0 (31 to 24)	Areg_0 (39 to 32)	Areg_0 (47 to 40)	0x10
Areg_1 (39 to 32)	Areg_1 (47 to 40)	Areg_0 (7 to 0)	Areg_0 (15 to 8)	0x14
Areg_1 (7 to 0)	Areg_1 (15 to 8)	Areg_1 (23 to 16)	Areg_1 (31 to 24)	0x18
Areg_2 (23 to 16)	Areg_2 (31 to 24)	Areg_2 (39 to 32)	Areg_2 (47 to 40)	0x1C
Areg_3 (39 to 32)	Areg_3 (47 to 40)	Areg_2 (7 to 0)	Areg_2 (15 to 8)	0x20
Areg_3 (7 to 0)	Areg_3 (15 to 8)	Areg_3 (23 to 16)	Areg_3 (31 to 24)	0x24
HASH1				0x28
HASH2				0x2C
Tx underrun	Good Tx frames			0x30
Rx overrun	Good Rx frames			0x34
Code error frames	CRC error frames	Deferred Tx frames		0x38
Single collision Tx frames		Multicollision Tx frames		0x3C
Acommit	Carrier loss errors	Late collisions	Excessive collisions	0x40
MaxRx		BSIZEreg	LEDreg	0x44

The internal registers are used less often than the host registers. They are used for:

- ❑ Setting diagnostic options, such as loopback (wrap) and copy all frames
- ❑ Setting network options. This is usually a one-time operation at initialization.

- Setting commit levels and PCI burst levels
- Interfacing via the management interface to the PHY registers
- Determining status interrupts
- Setting eight bytes of default PCI configuration data if the EEPROM checksum is bad
- Setting the various unicast and multicast addresses
- Providing network statistics
- Setting the LEDs and implementing a BIOS ROM

The NetCmd register is used to set many of the diagnostic modes such as wrap, copy short frames (CSF), copy all frames (CAF), no broadcast (NOBRX), duplex, and token ring frame formats. It also includes a reset bit, which is used to allow changes in the NetConfig register for additional network configuration options.

NetSio, the network serial I/O register**,** is used to control the MDIO/MDCLK management interface. It is also used to communicate with an EEPROM, using the EDIO/EDCLK serial interface. This register can also enable or disable PHY interrupts.

The NetSts and NetMask registers work in tandem to determine the nature of a status interrupt. The bits in the NetMask register are used to mask whether the status flags in NetSts cause interrupts or not.

The NetConfig register sets network configuration options during reset. This register can only be written to when ThunderLAN is in reset (NRESET = 0). It allows the controller to receive CRCs (RxCRC), pass errored frames (PEF), use a one-fragment list on receive (refer to subsection 5.3, One-Fragment Mode, for more information), use a single transmit channel, enable the internal PHY, and select the network protocol (CSMA/CD or demand priority).

The AREG registers allow ThunderLAN to recognize any four 48-bit IEEE 802 address. This includes specific, group, local, or universal addresses. They can be Ethernet or token ring addresses. The HASH registers allow group addressed frames to be accepted on the basis of a hashing table.

The statistics registers hold the appropriate network counters, including good Tx and Rx frames, collisions, deferred frames, and error counters. The LEDs are controlled through the LEDreg register, which directly controls the values output on the LED lines EAD[7::0]**.** All are, therefore, software programmable. LEDreg can also be used to implement a BIOS ROM. The Acommit register

is used to set the network transmit commit level. The BSIZEreg register is used to set the bus burst size on both Tx and Rx frames.

The internal registers are accessed via the DIO_DATA and DIO_ADR host registers. DIO_ADR holds the DIO address of the register. The data is then read from or written to DIO_DATA.

Before one can write to an internal register, one must find the proper address for the host registers to use as pointers to the internal register block, and decide whether to use the memory pointer or the I/O port pointer value. Following is an example of x86 C code to access a byte from an internal register using the I/O port pointer value:

```
//-------------------------------------------------------
// DioRdByte() – Read byte from adapter internal register
//
// Parameters:
// base_addr WORD base address of TLAN internal registers
// addr      WORD offset of register to read
//
// Return val:
// BYTE      value read
//-------------------------------------------------------
BYTE DioRdByte(WORD base_addr, WORD addr)
{
    outpw(base_addr+OFF_DIO_ADDR, addr);
    return(inp((base_addr+OFF_DIO_DATA) + (addr&3)));
}
```

The address of the register being read is determined by the calling program and is passed to this routine as a parameter, along with the the I/O base address. An output is executed to the DIO_ADR host register as part of setting up the pointer address. In x86 architectures, there are separate instructions for 16-bit port writes and 8-bit port writes; the 16-bit version is used to write all the address field's 16 bits in one operation. Internally, this causes the data from the internal register at that address to be deposited in the DIO_DATA host register. A byte read of the data register gets the LSbyte (addr&3). A more sophisticated routine honors the address of the byte specifically requested and sees that those eight bits are shifted down to a byte to be returned. If you want to read a whole word from one of the internal registers (32 bits), you could perform two 16-bit reads and merge the values to be returned as a 32-bit value like this:

```
//-------------------------------------------------------
// DioRdDword() – read 32 bits from internal TLAN register
//
//  Parameters:
//  base_addr WORD base address of TLAN internal registers
//   addr WORD address to read
//
//  Return val:
//   DWORD value read
//-------------------------------------------------------
DWORD DioRdDword(WORD base_addr, WORD addr)
{
 DWORD data;
 addr &= 0x3fff;
 outpw(base_addr+OFF_DIO_ADDR, addr);
 data = ((DWORD)inpw(base_addr+OFF_DIO_DATA))&0x0000ffffl;
 data |= ((DWORD)inpw(base_addr+OFF_DIO_DATA+2)) << 16l;
 return(data);
}
```

2.5 MII PHY Registers

Figure 2–6. MII PHY Registers

	Register	Description
Control	0x00	PHY generic control register
Status	0x01	PHY generic status register
PHY identifier	0x02	PHY generic identifier (high)
PHY identifier	0x03	PHY generic identifier (low)
AN advertisement	0x04	Autonegotiation advertisement
AN link-partner ability	0x05	Autonegotiation link-partner ability
AN expansion	0x06	Autonegotiation expansion
AN next page transmit	0x07	Autonegotiation next page transmit
Reserved	0x08	
Reserved	through	Reserved by IEEE 802.3
Reserved	0x0F	
Vendor-specific registers	0x10	
Vendor-specific registers	through	Vendor-specific registers
Vendor-specific registers	0x1F	

The 802.3 standard specifies a basic set of registers that must be present. These include a control register at 0x00 and a status register at 0x01. An extended set from 0x02 through 0x1F can also be implemented. Within the extended set, 0x02 through 0x07 are defined and 0x08 through 0x0F are reserved. The area between 0x10 and 0x1F can be used for vendor-specific applications. The basic set of registers is shown in dark gray above. The extended registers are shown in lighter gray and white. The light gray registers are those defined by 802.3, and the white registers are vendor-specific. The register specification for the internal 10Base-T PHY in ThunderLAN controllers is shown in Appendix A. We have also included the register specification for the TNETE211 100VG-AnyLAN physical media interface (PMI) in Appendix B.

The PHY registers are accessed to:

- ❑ Initialize the PHY, bring it in and out of reset, and isolate it from the MII
- ❑ Set PHY options, such as duplex and loopback
- ❑ Determine the type of speed and protocol supported by the PHY
- ❑ Conduct autonegotiation, if supported
- ❑ Select any vendor-specific options

The control register (GEN_ctl in ThunderLAN products) controls PHY options such as reset, loopback, duplex, and autonegotiation enable. It also powers down and isolates the PHY from the MII.

The status register (GEN_sts in ThunderLAN products) includes bits to identify the technology supported by the PHY. This technology includes protocol and duplex abilities. It indicates link, jabber, and autoconfiguration completion. Bit 0 of the status register also indicates whether the extended register set is supported.

The PHY identifier registers (GEN_id_hi/GEN_id_lo in ThunderLAN products) contain an identifier code for the silicon revision and the silicon manufacturer.

Registers 0x04 thru 0x07 are used in the autonegotiation process. They include the autonegotiation advertisement, autonegotiation link partner ability, autonegotiation expansion, and autonegotiation next page registers (AN_adv, AN_lpa, AN_exp respectively).

In the vendor-specific area, Texas Instruments has implemented a TLPHY_id register. This register is used to identify ThunderLAN-specific PHY devices. ThunderLAN also implements a specific control register, TLPHY_ctl, and status register, TLPHY_sts. The particulars of these registers change from PHY to PHY. Please refer to Appendix A for the PHY that you are using.

Writing to a register in a PHY through the management interface involves writing data and clock bits into NetSio**,** an internal register, which uses the pointer host registers. The data unit to or from a PHY register is always 16 bits.

The NetSio register uses three bits to drive the MDIO/MDCLK MII management interface. These bits are MCLK, MTXEN**,** and MDATA. These bits directly control the voltages present in the management interface and function like this:

- MCLK directly controls the MDCLK signal. Setting MCLK in NetSio high causes a logic 1 to appear on the MDCLK pin. Setting MCLK in the NetSio register low causes a logic 0 to appear on the MDCLK pin.
- MTXEN controls the direction of the MDIO pin.
  - When MTXEN is high, MDIO is driven with the value written on MDATA.
  - When MTXEN is low, MDATA mirrors the MDIO line.

Multiple PHYs can be attached to one MII. PHYs are selected through an address which can be in a range from 0x00 to 0x1F. Some vendors' PHYs have pins that can be pulled up or down to indicate the PHY address. In order for a particular PHY to be addressed, the driver must know the PHY address beforehand.

ThunderLAN's internal PHY for 10Base-T can only support two addresses. When used in conjunction with the rest of the ThunderLAN device, the address

is 0x1F. When the internal PHY for 10Base-T is used in a standalone mode, that is, when run from another controller through the MII pins, it is at address 0x00. These are the only two addresses allowed for the internal PHY.

The 100VG-AnyLAN PMI device, the TNETE211, is used to attach 802.12 physical media dependent (PMD) devices to ThunderLAN's MII. The TNETE211 has five external pins (DEVSEL[4::0]) that program the address to which it will respond. If multiple PHYs are used, each must be installed with a unique address.

Before reading or writing to any PHY register, the MII serial interface must be synchronized. This involves a one-time write of 32, 1 bits on the MDIO pin. Once this is done, an access can be done with a two-bit start delimiter, then a two-bit op code (for read or write), followed by five bits of PHY address, five bits of register address, two bits of turnaround time in case the PHY is going to write to the data line, and 16 bits of data.

The synchronization code could be done this way:

```
//-------------------------------------------------------
// MIISync() – send MII synchronization pattern to all
//              possible MII interfaces
//
//  Parameters:
//   base_addr   base address on TLAN internal registers
//
//  Return val:
//   none
//-------------------------------------------------------
void MIISync(WORD base_addr)
{
    register int i;
    clr(MTXEN);
```

where clr is a macro to set a bit to 0 in the NetSio internal register. In this case, bit MTXEN in NetSio is cleared.

```
#define clr(x)
DioWrByte(base_addr,Net_Sio,(BYTE)(DioRd
Byte(base_addr,Net_Sio)&~x))
```

When the output enable bit is cleared and the PHYs have just been turned on, none of them outputs data. The value on the data line is determined by the pull-

up resistor, which is recommended to be attached to this line. The MII devices should see 1s.

An alternate way to give the PHYs a series of 1s, is to:

```
set(MDATA)
set(MTXEN)
clr(MCLK);
//delay here
DioRdByte(base_addr,Net_Sio);
set(MCLK);
```

Where MCLK is a constant for the third LSB (in the internal NetSio register) and is defined as:

```
//delay
DioRdByte(base_addr,Net_Sio);
set(NMRST);
```

This is the command to set a bit to 0 in the internal NetSio register. In this case, the MCLK bit in NetSio is set. Set could be defined this way:

```
#define set(x)
DioWrByte(base_addr,Net_Sio,(BYTE)(DioRdByte
(base_addr,Net_Sio) |x))
```

The routine to synchronize the PHYs is part of the startup code. The controller at this point is held in reset due to the drivers writing a 1 to the Ad_Rst bit, bit 15 in the HOST_CMD register, or a reset being received on power-up through the PCI system. Setting the NMRST bit to 0 places the MII bus in a reset state.

```
for (i = 0;i < 32;i++)
   togLH(MCLK);
```

The command togLH is a combination of the clr and set commands on the passed parameter and is defined this way:

```
#define togLH(x)   {clr(x); \
                        set(x);}
}
```

togLH is repeated 32 times to give PHYs the 32, 1 data bits that they need to get synchronized. Note that the clock line is left in the high state at the end of the loop.

After synchronization, one could use code like the following to read a PHY register:

```
//-------------------------------------------------------
//  MiiRdWord() – Read word from Phy MII, place at ptr,
return status
//
//  Parameters:
//   base_addr  WORD  base address of TLAN internal regis-
ters (passed
//                    for set/clr macros)
//   dev        WORD  device to read from
//   addr       WORD  register on dev to read from
//   pval       WORD* storage for data read
//
//  Return val:
//   int        OK (0) on success, NO_ACK (1) on failure
//-------------------------------------------------------
int MiiRdWord(WORD base_addr, WORD dev, WORD addr, WORD
*pval)
{
   // mread array: 01 is the start delimiter sequence for
   the MII
   // interface, 10 specifies the operation will be a
   read.
   // See IEEE 802.3u
   WORD i,tmp;
   char ack;
   BYTE b;
   WORD diodata = base_addr+OFF_DIO_DATA+Net_Sio;
   CritOn();
   outpw(base_addr+OFF_DIO_ADDR,Net_Sio);
```

This example uses the host registers as I/O ports, so the code needs to resolve a pointer to the host NetSio register. NetSio is added to the base DIO_DATA register address to give the byte offset. The NetSio register is only one byte long, and a byte port read instruction needs to activate the proper byte strobe in the PCI interface. Filling in the lowest two bits with the NetSio offset constant causes the NetCmd register to be read.

Interrupts are turned off with the CritOn() macro. This macro leaves a value that can be sampled to see if it has been invoked. CritOn can be defined as follows:

```
#define CritOn()  if (CritLevel == 0) \
                  { _asm { cli } }    \
                  CritLevel++
```

The NetSio register must be reached indirectly using the host registers. This sets the address of the NetSio register, offset from the beginning of the internal register block, in the host register which is used as an address pointer to the Internal registers. This is a two-active-byte-strobe (out of four) write cycle. The DIO_ADR register is 16 bits in width.

```
b = inp(diodata);
b &= ~MINTEN;
b &= ~MDATA;
b |= MCLK;
outp(diodata,b);
```

This cycle reads, modifies, and writes the contents of the NetSio register. It turns off the MII interrupt by forcing the MINTEN bit to a logic low, makes sure the data bit in the interface comes on with a logic low when enabled in the next write, and makes sure the clock line in the two-wire MII management interface starts high.

```
b |= MTXEN;
outp(diodata,b);
```

The previous code turns on the data output driver. ThunderLAN has to write several fields to the MII before data is passed in either direction.

```
//togLH
b &= ~MCLK; outp(diodata,b);
b |= MCLK; outp(diodata,b);         //0 data bit out
```

This samples data on the rising edge of the MCLK bit. Take the first bit into the PHY MII as follows:

```
b &= ~MCLK; outp(diodata,b);
b |= MDATA; outp(diodata,b);
b |= MCLK;  outp(diodata,b);        //1 data bit out
```

This concludes writing out the start delimiter bits. The data can be changed before the clock is taken low, as when shifting out the operation code as follows:

```
b |= MDATA; outp(diodata,b);  //1st part not nec.
//togLH
b &= ~MCLK; outp(diodata,b);
b |= MCLK; outp(diodata,b);         //1
b &= ~MDATA; outp(diodata,b);
//togLH
b &= ~MCLK; outp(diodata,b);
b |= MCLK; outp(diodata,b);         //0
```

10 is the read op code for an MII management operation.

```
// Send the device number Internal=31(0x1f),
   External=0(0x00)
for (i = 0x10;i;i >>= 1) /*  10 is the read op code*/
{
    if (i&dev)
        b |= MDATA;
    else
        b &= ~MDATA;

    outp(diodata,b);
    //togLH
    b &= ~MCLK; outp(diodata,b);
    b |= MCLK; outp(diodata,b);
}
```

The following loop index is used as a mask to walk through the device number which is passed to this routine as a parameter. Each loop looks at a bit in the device number, starting with the MSB. It sets the MDATA bit to match the internal representation of the NetSio register before outputting the composite value

to NetSio. Then the clock is cycled for each bit. The loop effectively cycles five times.

```
// Send the register number MSB first
// Send the device number Internal=31(0x1f),
   External=0(0x00)
for (i = 0x10;i;i >>= 1)
{
    if (i&addr)
        b |= MDATA;
    else
        b &= ~MDATA;
    outp(diodata,b);
    //togLH
    b &= ~MCLK; outp(diodata,b);
    b |= MCLK; outp(diodata,b);
}
// 802.3u specifies an idle bit time after the register
// address is sent. This and the following zero bit are
// designated as ”Turn–around” cycles.
b &= ~MTXEN; outp(diodata,b);
```

To get an idle bit, turn off the data driver, then cycle the clock.

```
//togLH
b &= ~MCLK; outp(diodata,b);   //end turn around cycle
b |= MCLK; outp(diodata,b);    //this should clock “0”
                              ackn bit out
b &= ~MCLK; outp(diodata,b);   //take clock low wait
                              for data valid
```

After the addresses have been clocked out on a read cycle, there is a cycle where neither side drives the data pin. If the PHY is synced and ready to respond, it should drive a 0 next, followed by the 16 bits of data. The data is available up to 300 ns after the rising edge of the clock, so the software loop uses that time to execute the instruction to make the clock go low again.

```
// Get PHY Ack
ack = inp(diodata);
if (!(ack & MDATA))      // if ack=0, record bits
{
    b |= MCLK; outp(diodata,b);   // complete ack
                                  cycle clock
    for (tmp = 0,i = 0x8000;i;i >>= 1)
```

The loop is set for 16 cycles, using the loop variable as a mask for pointing to the bit position stored. The MSB comes in first. For each shift cycle, the clock goes up to start the access and goes down to guarantee that some time elapses between the rising edge of the clock and the time the data is sampled.

```
    {
      b &= ~MCLK; outp(diodata,b);
      if (inp(diodata)&MDATA)
            tmp |= i;   //if data bit=1, or position in
       b |= MCLK; outp(diodata,b);
    }
}
else
```

If the PHY does not respond, one needs to complete the access cycle to keep other PHYs from being left in mid-access. Leave the MDIO pin set for input but set the data variable to all 1s. This routine gives 17 clock cycles, using the macro for togHL on the MCLK bit of the NetSio register. There are 17 clock cycles, because the first one finishes the acknowledge cycle (the clock was left in a logic low state when the data was read).

```
{
    for (i = 0;i < 17;i++)
        togLH(MCLK);
    tmp = 0xffff;
}
//togLH
b &= ~MCLK; outp(diodata,b);
b |= MCLK; outp(diodata,b);
b = inp(diodata);
```

This is the quiescent cycle following data transmission. Since this is a read operation, ThunderLAN does not drive the line and the PHY turns off during this cycle. If the quiescent cycle is not performed between the read and write operations, the PHY is not able to assert the MDIO pin low to indicate a PHY interrupt. After this cycle and a read, the driver sets the MINTEN bit high, which enables PHY interrupts.

```
set(MINTEN);
*pval = tmp;
CritOff();
```

The function value returned is reserved for completion and error codes, and is returned via a pointer. CritOff turns on the interrupts again and is defined as:

```
#define CritOff() if (--CritLevel == 0) \
                  { _asm { sti } }
```

A similar routine with similar code is used to write values into the PHY registers through the management interface.

2.6 External Devices

This following section discusses the manner in which the ThunderLAN controller interfaces to external devices. These devices include:

- A BIOS ROM
- Light emitting diodes (LEDs)
- A serial EEPROM
- Any devices (PMIs/PMDs) attached to the MDIO/MDCLK serial interface of the MII

2.6.1 BIOS ROM

A BIOS ROM is supported with two external latches and a memory device. A memory-space base register is implemented at location 0x30h in the PCI configuration space, with 16 bits forced to fixed values. This reserves 64K bytes of memory space. A PCI memory read is requested, and if the upper 16 bits match the value posted in the BIOS ROM base address, hardware state machines begin a special cycle that posts the two eight-bit parts of the address along with address strobes on the EAD[7::0] pins. The EAD[7::0] lines act as an output bus during the output of the low eight bits signaled by the EALE strobe, and the output of the next eight bits signaled by the EXLE strobe. They act as an input bus when accepting the data from the EPROM which is signaled by the EOE EPROM output enable strobe. This interface is designed to support all types of read cycles from the host: byte, word, and long word. Four cycles are automatically done to prepare a 32-bit response to the PCI read cycle. During the state machine's execution, the PCI read cycle sends wait states to the host processor. Writes to the EPROM memory space are accepted and performed, but are internally ignored.

2.6.2 LEDs

The EAD[7::0] bus is an output bus when not involved in EPROM read cycles. These pins are driven with the inverse of the pattern written into LEDreg, an internal register. To access this register, a 0x44 is written to the DIO_ADR host register, then either a byte write to the DIO_DATA host register or a read/modify/write to the whole DIO_DATA host register is done to deposit the value into LEDreg. A logic 1 in the register translates to an active low on the external output pin. All bits in this register are set to 0 on the Ad_Rst bit**,** or when the external reset, PRST#, is activated.

The meaning assigned to the LEDs, which LEDs are actually implemented, and the times to set and clear them are all programmable. Texas Instruments

reserves the following two LED locations for its drivers. The bit numbers refer to their locations in LEDreg.

- Bit 0 (LSB) displays link status.
- Bit 4 displays activity.

2.6.3 EEPROM

The implementation-specific configuration information is read or written into the EEPROM from two sources. Control of the two-wire serial bus to the EEPROM (EDIO and EDCLK bits) on reset (hardware or software) rests with the four bits in the PCI_ NVRAM register (DATA, DDIR, CLOCK, CDIR) in the PCI configuration space. Any time this register is written to, control of the EDIO/EDCLK bus reverts back to this register.

The other possible source of values for this bus is from an internal register, the network serial I/O register NetSio. Here the three bits used to control the interface are EDATA, ETXEN, and ECLOK. The PCI_ NVRAM register interface to this external EEPROM port was designed assuming that there might be another master device on this bus. Note that NetSio does not implement a clock direction register. Assuming that only one EEPROM is on the serial bus and only the ThunderLAN device is driving the bus, both control implementations are equivalent. Use NetSio when possible to read or write to the EEPROM.

Writing to the NetSio register involves writing a >000 to the host register DIO_ADR, then writing to the DIO_DATA host register. Control of the EEPROM interface shifts to the bits in NetSio when a write takes place to the DIO_DATA host register. Following is an example of how one might read a byte of data from the EEPROM, using the control bits in NetSio from the internal register block.

```
//-------------------------------------------------------
//  EeRdByte() –    read byte of data from EEPROM (see
                Exel XL24C02 device specification)
//
//  Parameters:
//   base_addr      WORD base address of TLAN internal
                registers
//   addr           WORD address to read
//
//  Return val:
//   BYTE           value read
//-------------------------------------------------------
BYTE EeRdByte(WORD base_addr, WORD addr)
   {
    int i,ips,tmp;


    CritOn();
```

CritOn turns off the interrupts. Remember that there are two possible control points for reading and writing to the EEPROM. This is an attempt to avoid a control shift and avoid loss of focus on just which byte, word, or double word one accesses, since accessing the EEPROM is a relatively long process.

```
   // send EEPROM start sequence
   set(ECLOK); set(EDATA); set(ETXEN); clr(EDATA);
   clr(ECLOK);
```

Set and clear are macros for a read/modify/write routine for individual bits in the NetSio register. The NetSio byte is read indirectly from the internal register block with the host register address and data pointers, the bit passed as a constant (really a bit mask) is ANDed to 0 (clear), or ORed to a 1 (set). The pattern of bits to be set and cleared is given in the data sheet for the EEPROM.

```
// put EEPROM device identifier, address, and write
// command on bus
sel(base_addr, WRITE);
```

The EEPROM, with its serial interface, must receive a wakeup pattern and a device number, since more than one device can be tied to this bus. This code assumes that there is only one device, 0.

```
// EEPROM should have acked
if (!ack(base_addr))
    return (0);
// send address on EEPROM to read from
sendaddr(base_addr, addr);
```

sendaddr is a routine for serially sending out the address to be read from the EEPROM. Each bit of the address must be accompanied by a clock.

```
// EEPROM should have ack'd address received
if (!ack(base_addr))
    return (0);


// send EEPROM start access sequence
set(ETXEN); set(ECLOK); set(EDATA); set(ETXEN);
clr(EDATA); clr(ECLOK);
```

When the EEPROM address is shipped out, another special pattern of control signal movements must take place to signal the start of the data transfer.

```
// send device ID, address and read command to EEPROM
sel(base_addr, READ);
// EEPROM should have acked
if (!ack(base_addr))
    return (0);
//clock bits in from EDATA and construct data in tmp
for (i = 8,tmp = 0,ips = 0x80;i;i--,ips >>= 1)
{
    set(ECLOK);
    if (test(EDATA))
        tmp |= ips;
    clr(ECLOK);
}
togHL(ECLOK);
```

Test is a macro, like set and clear, that indirectly reads the bit passed into the NetSio register in the internal register block using the DIO_ADR and DIO_DATA registers in the host register block.

```
// send EEPROM stop access sequence
clr(EDATA); set(ECLOK); set(EDATA); clr(ETXEN);
```

The following control signal movements are specified in the data sheet for the EEPROM.

```
CritOff();
return((BYTE)(tmp & 0xff));
}
```

Similar routines can be created for writing a byte, reading and writing a word, or reading and writing a double word to the EEPROM using the NetSio register's control bits. In the DOS/Windows environment, there are O/S calls provided for reading and writing to PCI configuration space. Routines similar to network serial I/O routines can be written using the PCI O/S calls, but they are more awkward. Instead of an indirect read, modify, and write cycle, one performs an O/S PCI read call, a modify, and an O/S PCI write call for each bit modified.

2.6.4 ThunderLAN EEPROM Map

ThunderLAN uses the following EEPROM map. Note that these values may be used in several applications and systems including:

- ThunderLAN hardware
- A host running Texas Instruments ThunderLAN drivers
- Texas Instruments diagnostic routines

Table 2–1. ThunderLAN EEPROM Map

Address	Default	Binary Bits	Description
0x70	0x70	ccccxbdx	Acommit register and bit level PHY controls cccc–commit level 0–7 b–Local loopback select d–ThinNet select
0x71	0x33	ttttrrrr	Transmit and receive burst size control tttt–Transmit burst size control 0–7 rrrr–Receive burst size control 0–7
0x72	0x00	ccccbbbb	PHY TLPHY_ctl register initialization options cccc–Bits 15–12 of the MII register 0x11 TLPHY_ctl bbbb–Bits 3–0 of the MII register 0x11 TLPHY_ctl
0x73	0x0f		Interrupt pacing timer value
0x74	0xff		Configuration space Latency_Timer register value
0x75	0xea		LSB of maximum frame size to test
0x76	0x05		MSB of maximum frame size to test
0x77	0x40		LSB of small frame in mixed frame size test
0x78	0x00		MSB of small frame in mixed frame size test

Table 2–1. ThunderLAN EEPROM Map (Continued)

Address	Default	Binary Bits	Description
0x79	0x04	WxSHRAFI	PHY and test control modes W – PHY wrap request S – Skip training request H – HiPriority transmit frames request R – Don't copy short frame request A – Don't copy all frames request F – Full duplex request I – Internal ThunderLAN wrap request
0x7a	0x02	LFxTxADP	Check modes and frame type L – Ignore last byte plus one DMA no write test F – Zero (ignore) bits 12 and 13 of Rx length request T – Token ring format request A – AT&T2X01 fix enable (receive logic state machine tweak) D – Stop on first error P – PMI wrap length check disable request
0x7b	0x05		Test to execute 1 – Multicast test 2 – Pipe test 3 – Mix pipe test 4 – PMI/TI-PHY interrupt test 5 – Frame read/write test 6 – Adapter check test
0x7c	0x14		Pipe depth value
0x7d	0x0a		Reserved
0x7e	0x00		LSB of NetConfig register value
0x7f	0x06		MSB of NetConfig register value Note: Multichannel mode must be selected for high priority frames to be sent
0x80	0x82		
0x81	0x08		

Table 2–1. ThunderLAN EEPROM Map (Continued)

Address	Default	Binary Bits	Description
0x82	0x00		
0x83			Ethernet address
0x84			Ethernet address
0x85			Ethernet address
0x87			Ethernet address
0x88			Ethernet address
0x86			Ethernet address
0x89	0xff		Checksum
0x8a	0xff		Checksum
0x8b	0x83		
0x8c	0x08		
0x8d	0x00		
0x8e			Token ring address
0x8f			Token ring address
0x90			Token ring address
0x91			Token ring address
0x92			Token ring address
0x93			Token ring address
0x94	0xff		Checksum
0x95	0xff		Checksum
0x96	0x82		
0x97	0x08		
0x98	0x00		
0x99			Ethernet address
0x9a			Ethernet address
0x9b			Ethernet address

Table 2–1. ThunderLAN EEPROM Map (Continued)

Address	Default	Binary Bits	Description
0x9c			Ethernet address
0x9d			Ethernet address
0x9e			Ethernet address
0x9f	0xff		Checksum
0xa0	0xff		Checksum
0xa1	0x83		
0xa2	0x08		
0xa3	0x00		
0xa4			Token ring address
0xa5			Token ring address
0xa6			Token ring address
0xa7			Token ring address
0xa8			Token ring address
0xa9			Token ring address
0xaa	0xff		Checksum
0xab	0xff		Checksum
0xac	0x82		
0xad	0x08		
0xae	0x00		
0xaf			Ethernet address
0xb0			Ethernet address
0xb1			Ethernet address
0xb2			Ethernet address
0xb3			Ethernet address
0xb4			Ethernet address
0xb5	0xff		Checksum

Table 2–1. ThunderLAN EEPROM Map (Continued)

Address	Default	Binary Bits	Description
0xb6	0xff		Checksum
0xb7			
0xb8			
0xb9			
0xba			
0xbb			
0xbc			
0xbd			
0xbe			
0xbf			
0xc0			Vendor ID register LSbyte
0xc1			Vendor ID MSbyte
0xc2			Device ID LSbyte
0xc3			Device ID MSbyte
0xc4			Revision
0xc5			Subclass
0xc6			Min_Gnt
0xc7			Max_Lat
0xc8			Checksum

Chapter 3

Initializing and Resetting

This chapter describes the steps necessary to get a ThunderLAN device ready to transmit and receive frames. It provides examples of the necessary code, beginning with configuration of the ThunderLAN device on a peripheral component interconnect (PCI) system. The chapter also defines the steps needed for both hardware and software reset.

Topic **Page**

3.1 Initializing

To initialize or to set the starting values for ThunderLAN, the device must proceed through a specific sequence of steps. This procedure assumes that the autoconfiguration step of loading from the EEPROM to the PCI configuration registers has already taken place.

3.1.1 Finding the Network Interface Card (NIC)

A PCI BIOS call must be performed to determine if there is a PCI card present with a ThunderLAN controller. A ThunderLAN controller should have a ThunderLAN device ID and should also have a vendor ID. The example code uses the TI vendor ID. The call to find the board is:

```
#define TLAN_DEVICEID   0x0500  //PCI TLAN device ID)

...

#define TI_VENDORID     0x104C  //PCI vendor ID
assigned to TI

...

if (PciFindDevice(TLAN_DEVICEID, TI_VENDORID, 0,
   &nic.DevId))

   error(”The PCI Bios can’t find a TLAN board”);
```

PciFindDevice is further broken down to an O/S call to the PCI interrupt service routines in the BIOS formatted as:

```
//-------------------------------------------------------

//  PCIFindDevice – Find PCI device

//

//  Parameters:

//   DeviceID       WORD The device ID

//   VendorID       WORD The vendor ID

//   Index          WORD index (normally 0, use when more
than 1 device)

//   pDev           WORD* Where to put the device id

//

//  Return val:

//   int            0 if successful. See std return codes
in header

//-------------------------------------------------------
```

```
WORD PciFindDevice(
    WORD    DeviceID,
    WORD    VendorID,
    WORD    Index,
    WORD    *pDev)
{
    union REGS r;
    r.h.ah = PCI_FUNCTION_ID;
    r.h.al = FIND_PCI_DEVICE;
    r.x.cx = DeviceID;
    r.x.dx = VendorID;
    r.x.si = Index;
    int86(PCI_INT, &r, &r);
    *pDev = (WORD)r.x.bx;
    return (int)r.h.ah;
}
```

When the BIOS call is finished, the value returned is 0 if successful or an error code if not successful. Once the BIOS board is found, references to it and properties assigned to it by the O/S are indirectly referenced by the value returned to nic.devid. The structure nic is a collection of properties belonging to the NIC. As the sample code learns more about the environment with respect to the NIC, other information in this structure is filled in.

3.1.2 Finding the Controller in Memory and I/O Space

To access the host registers, the I/O base address must be determined. This I/O base is needed, since the host registers are accessed as I/O ports. The I/O base address register in the ThunderLAN controller has the LSB hardwired to high. This code does an O/S call to read a 32-bit word from PCI_MEMBASELO in the configuration space belonging to this board's PCI device ID. If the first base register is not an I/O register, the second base register location is checked. If an I/O base register is found, it is stored away in the structure nic. If neither of the first two locations is a valid I/O base register, an error is declared and the program is ended. Note that the configuration space was originally supplied with a space request and the operating system as part of the power-on self test (POST) supplied the card with sufficient address space by filling in the RAM bits in the base registers.

```
#define PCI_MEMBASELO    0x10   //low memory base
address register
#define PCI_IOBASELO     0x14   //low I/O base address
register
...
nic.IoBase = PciRdWord(nic.DevId,PCI_MEMBASELO);
    if (!(nic.IoBase & 1))
    {
        nic.IoBase = PciRdWord(nic.DevId,PCI_IOBASELO);
        if (!(nic.IoBase & 1))
            error(”PCI Config failed: Unexpected
non-I/O address found”);
    }
```

3.1.3 Finding Which Interrupt was Assigned

When the base register is established, the driver needs to find out what interrupt was assigned to the card. The next code segment from GetPciConfig below retrieves the PCI_INTLINE which in x86-based PCs refers to the interrupt request (IRQ) numbers (0–15) of the standard dual 8259 configuration. Note that this piece of information is retrieved via the key parameter of the evaluation module network interface card's (EVMNIC's) PCI devIce ID.

```
   (#define PCI_INTLINE 0x3C)
...
   if (!(nic.Irq = PciRdByte(nic.DevId, PCI_INTLINE)))
       error(”PCI Config failed: Unconfigured interrupt”);
```

Implemented hardware interrupts in a PC range from 0 to 15, but 0 is usually unavailable to peripherals. It is suggested that a value of 0 or greater than 15 be rejected. This gives greater system protection over a check for 0 or 255, which is the PCI-compliant answer for none available. (Refer to the *PCI Local Bus Specification*, section 6.2.4.)

3.1.4 Turning on the I/O Port and Memory Address Decode

The next step in the GetPciConfig section of the code is responsible for turning on the ThunderLAN controller by enabling the decode of memory and I/O port addresses. Without this step, there is no access to the host registers and therefore, to the internal registers or the MII granted to the host processor.

The PCI specification calls for the shutdown of address decode in both I/O port space and memory space upon PCI reset to avoid multiple devices responding to bus cycles before the operating system has a chance to sort out space requirements. ThunderLAN complies with this requirement. Configuration space access is not shut down on reset, as each slot has a chip select line guaranteeing unambiguous accesses.

```
//  Enable I/O and memory accesses to the adapter, and
 bus master

PciWrWord(nic.DevId, PCI_COMMAND, IO_EN | MEM_EN |
BM_EN);
```

Where these constants have the following values:

```
#define   PCI_COMMAND  0x04

#define   IO_EN        0x0001

#define   MEM_EN       0x0002

#define   BM_EN        0x0004
```

and PciWrWord, a register level int86 O/S call, has the following definition:

```
void PciWrWord(WORD devid, WORD addr, WORD data).
```

The PciWrWord statement goes to the PCI configuration space associated with the evaluation module (EVM) device ID and writes to the PCI command register. This sets the three LSBs to enable I/O map decodes, memory decodes, and allows bus mastering to occur via the NIC.

Of the two signals, IO_EN and MEM_EN, the driver only needs to activate the mechanism that is used to address the ThunderLAN controller: either I/O ports or memory. Both are activated here in the sample code. BM_EN is required by the ThunderLAN device to operate properly. All the network data must be moved to and from the host by a ThunderLAN-initiated direct memory access (DMA), but this capability must be separately enabled, as required by the *PCI Local Bus Specification.* In other words, the PCI configuration register must have all three of these bits and they must function in this way.

3.1.5 Recovering the Silicon Revision Value

At this point, the sample program needs to know what the default silicon revision for the controller is. There is a revision byte in the configuration space that can be read with a PciRdxxxx command. This configuration byte is loaded with EEPROM data to signal the board-level revision code. If the EEPROM data is bad or nonexistent, a value for this byte is hardwired in an internal register at location 0x0c. This byte indicates the silicon revision. However, once the memory and I/O access modes are turned on, one can read this register directly and get the silicon revision, regardless of whether this default value was needed in the PCI initialization. With this arrangement, the driver can find the silicon revision and the board revision.

```
nic.Rev = DioRdByte(nic.IoBase,NET_DEFREV);
...
#define NET_DEFREV 0x0C  //default revision reg
```

DioRdByte calls a routine that loads the host DIO_ADR register with NET_DEFREV and does a byte-enabled read of the host DIO_DATA register, returning the value to the member rev value of DIO_DATA for structure nic.

3.1.6 Setting the PCI Bus Latency Timer

An additional step that must be performed in the PCI configuration section of the code is to set the latency timer to the maximum value of 0xff. It had been loaded with 0 at reset. The instruction is:

```
PciWrByte(nic.DevId,PCI_LATENCYTIMER,0xFF);
...
#define PCI_LATENCYTIMER  0x0D
```

Where PciWrByte is a register-level interrupt 86 (int86) O/S call for reading PCI configuration space, nic.devId is the NIC's PCI system identifier discovered above, and PCI_LATENCYTIMER is a constant representing the offset into the PCI configuration space of that byte.

3.2 Resetting

Resetting ThunderLAN is required when conditions such as an incorrect power-up cause the register value in the device to deviate from that needed for proper operation. To perform either a software or hardware reset, the programmer must complete the steps indicated.

3.2.1 Hardware Reset

The IEEE 802.3 specification defines a power-up routine which must be followed to ensure that ThunderLAN's internal 10Base-T PHY powers up correctly. This routine also allows for the additional delay necessary when a crystal is used to drive the FXTL1 and FXTL2 lines.

1) Sync all attached PHYs
2) Isolate all attached PHYs by writing the PDOWN, LOOPBK, and ISOLATE bits into the control register (the GENen_ctl register in ThunderLAN)
3) Enable the internal PHY by writing 0x4000h (LOOPBK) in the GENen_ctl register
4) Wait 500 ms for the crystal frequency to stabilize
5) Reset the PHY state machine by asserting the LOOPBK and RESET bits in the GENen_ctl register
6) Resync the internal PHY
7) Read the control register until bit 15 (RESET) = 0 and the PHY comes out of reset. This is the time needed to read the register.
8) Load the selected PHY options into the GENen_ctl register
9) If using the attachment unit interface (AUI) mode, set the AUI bit in the TLPHY_ctl register
10) Wait one second for the clocks to start

3.2.2 Software Reset

The driver needs to reset ThunderLAN at startup when an adapter check interrupt occurs or when an upper layer requires the driver to do so. ThunderLAN may only need to be reinitialized when link is lost. To reset ThunderLAN the driver must:

1) Clear the statistics by reading the statistics registers
2) Issue a reset command to ThunderLAN by asserting the Ad_Rst bit in the HOST_CMD register
3) Disable interrupts by asserting the Ints_off bit in HOST_CMD
4) Setup the Areg and HASH registers for Rx address recognition
5) Setup the NetConfig register for the appropriate options
6) Setup the BSIZEreg register for the correct burst size
7) Setup the correct Tx commit level in the Acommit register
8) Load the appropriate interrupt pacing timer in Ld_Tmr in HOST_CMD
9) Load the appropriate Tx threshold value in Ld_Thr in HOST_CMD
10) Unreset the MII by asserting NMRST in NetSio
11) Initialize the PHY layer
12) Setup the network status mask register, NetMask
13) Reenable interrupts by asserting the Ints_on bit in HOST_CMD

Chapter 4

Interrupt Handling

ThunderLAN and its host device indicate communication with each other by sending and receiving interrupts to the bus data stream. This chapter provides information on setting up code which recognizes, prioritizes, and acknowledges these interrupts. It defines specific interrupt codes and describes what happens when these occur. This allows the user to diagnose and correct any conditions which may cause failure.

Topic **Page**

4.1 Loading and Unloading an Interrupt Service Routine (ISR)

Before the ThunderLAN controller can be allowed to generate an interrupt to the host, it is necessary to install code for the host to handle the interrupt. The driver also relies on other host services that are interrupt-driven, like getting notice of timer ticks for deadman timers. The driver calculates the pattern to write to the interrupt controller to acknowledge the interrupt from the controller, based on the actual hardware interrupt line assigned to the NIC's slot.

This sample program hooks into the software vector table. The host PC timer interrupt comes first, so that the program can time out operations that can hang the PC and can also provide time stamp information for operations.

```
nic.OldTimer = HwSetIntVector((BYTE)0x1C, TimerIsr);
```

The driver points to the next code to execute when the timer interrupt goes off and saves the value for program restart. nic.OldTimer is a storage place for this information reserved in a routine called NICEVN, which is allocated for each NIC; this instance is called nic. HwSetIntVector is a function that inserts the address of a function into the interrupt table at a particular location:

```
// HwSetIntVector() – Set PC interrupt vector and return
previous one
//--------------------------------------------------------
//  Parameters:
//   bIRQ           BYTE            Irq to set
//   lpFunc         void (ISR *)()  interrupt function
//
//  Return val:
//   void (ISR *)() Returns previous contents of vector
//--------------------------------------------------------
static void (ISR *HwSetIntVector( BYTE bIRQ, void (ISR
*lpFunc)() ))()
{
      BYTE bINT;
      void (ISR *lpOld)();
      if( bIRQ < 8 )
            bINT = 8 + bIRQ ;
      else if (bIRQ < 15)
            bINT = (0x70–8) + bIRQ;
      else
            bINT = bIRQ;
      lpOld = _dos_getvect( bINT );
            _dos_setvect( bINT,lpFunc);
      return( lpOld );
}
```

This routine converts either the eight low hardware interrupts, or the eight high interrupts, or a software interrupt higher than 0xF to the vector table, then makes an O/S call to get the old vector and slips in the new. It returns the previous contents of that table entry so that it can be restored later. The timer interrupt is not connected to one of the 15 hardware interrupts, so its vector is higher than 0xF and is entered explicitly.

The code called is:

```
void interrupt far TimerIsr()
{
    if (timercount)
        if (--timercount == 0)
            IndicateEvent(EV_TIMEOUT);
    if (nic.OldTimer)
        (*nic.OldTimer)();
}
```

If a time-out value has been set, nic.OldTimer is decremented. If it reaches 0, it sets an internal program flag that is checked in the main program loop. If the OldTimer value is not 0 (there was a vector saved in this NIC's instance of the structure NICEVN), the routine branches to that code so that whatever else needs to be done on the PC on a timer tick is done.

The next interrupt service routine to be intercepted is the abort routine. If the user tries to kill this program with a control break, the sample program makes an attempt to put back all of the interrupt vectors as they were found before exiting the program. The call to insert code from the sample program is similar to the timer intercept above:

```
nic.OldAbort = HwSetIntVector((BYTE)0x23, AbortIsr);
```

The code that gets executed is:

```
void interrupt far AbortIsr()
{
      cleanup();
      abort();
}
```

Cleanup uses the same HwSetIntVector routine to restore the old value. This time, the parameter is the old value and the interim value returned by the function is ignored. Only the three interrupts that were asserted are restored, and only if the structure for the NIC instance has had old values saved in it.

```
//-------------------------------------------------------
// cleanup() – cleanup in preparation for exiting to dos
//-------------------------------------------------------
static void cleanup(void)
{
      if (nic.Irq)
            HwSetIntMask((BYTE)nic.Irq,1);
      if (nic.OldNic)
            HwSetIntVector((BYTE)nic.Irq,nic.OldNic);
      if (nic.OldTimer) HwSetIntVector(0x1C,nic.OldTimer);
      if (nic.OldAbort) HwSetIntVector(0x23,nic.OldAbort);
}
```

The next interrupt to be intercepted corresponds to the actual hardware interrupt. Since the original installation for a ThunderLAN controller is a PCI slot, its interrupt in the PC is assigned by the PCI BIOS code. You must interrogate the PCI configuration space to find it. This was done in the previous section as part of GetPciConfig(). It uses the same function call as the two intercept installs below:

```
nic.OldNic = HwSetIntVector((BYTE)nic.Irq, NicIsr);
```

Note that the actual interrupt request (IRQ) is passed as a variable, not a constant as in the interrupts installed above. As in most interrupt service routines, make sure that stack space is available for calls and protect the information currently in the registers.

4.2 Prioritizing Adapter Interrupts

All (non-PCI) adapter interrupts are governed by the interrupt pacing timer. The interrupt pacing timer is started whenever the HOST_CMD register Ack bit is written as a 1. When this timer expires and if any interrupt sources are active, a PCI interrupt is asserted. When the host reads the HOST_INT register, the value it reads indicates the highest priority interrupt that is active at that time.

Interrupts are prioritized in the following order:

- Adapter check interrupts cause an internal ThunderLAN reset, clearing all other interrupt sources. Adapter check interrupts can only be cleared by a PCI hardware (PRST#) or software (Ad_Rst) reset.
- Network status interrupts
- Statistics interrupts
- List interrupts, which service transmit and receive interrupts in a round-robin fashion.
  - Receive end of frame (EOF) interrupts have higher priority than receive end of channel (EOC) interrupts (an Rx EOC cannot occur until all EOFs have been acknowledged).
  - Transmit interrupts are prioritized in channel order; channel 0 has lowest priority. Transmit EOF interrupts have higher priority than transmit EOC interrupts (a Tx EOC cannot occur until all EOFs have been acknowledged).

4.3 Acknowledging Interrupts (Acking)

The ThunderLAN controllers have been designed to minimize the code necessary to acknowledge interrupts. This is accomplished by matching the HOST_INT register's bits to the corresponding bits in the HOST_CMD register. Also, the HOST_INT's two LSBs are set to 0 so that it forms a table-offset vector, which can be used in a jump table. This allows for quick branching to the appropriate interrupt service routine.

To acknowledge interrupts:

- ❑ Disable all interrupts.
- ❑ Create a jump vector from the value read in HOST_INT.
- ❑ Use a jump table or a conditional branch structure to branch to the ISR.
- ❑ Execute the appropriate commands for the particular interrupt.
- ❑ Load the Ack_Count register with the number of interrupts to be acknowledged. This is useful if several EOF interrupts are acknowledged at once.
- ❑ Write to HOST_CMD. You may assert the GO bit (Ack and GO commands), if desired.

Interrupts can be disabled by writing to the HOST_INT register. One quick and easy way of doing this is by writing the contents of HOST_INT back after reading it at the start of the interrupt routine.

A jump table contains the starting address of the individual interrupt routines. Offsets to this table can be easily created from the HOST_INT vector read. It may be necessary to shift the vector read in order to factor out bits 1 and 0 (They are read as 0 always). Using this table or a conditional branching structure, the appropriate jump to the interrupt routine is easy to find.

The interrupt routine that is branched to performs the commands for the interrupt call. In some cases, this involves loading Ack_Count with a value greater than 1. This is particularly true when acknowledging Tx EOF with the Ld_Thr register loaded.

To acknowledge the interrupt, write the HOST_INT vector into HOST_CMD. The bit values in HOST_INT have a one-to-one correspondence with HOST_CMD. This simplifies the driver code and saves programming time. Interrupts should be reenabled when exiting the interrupt routine. Acknowledging interrupts to HOST_INT achieves this goal.

4.4 Interrupt Type Codes

The following subsections define specific interrupt codes which may occur during ThunderLAN operation. It describes the conditions that result from the occurrence of interrupts, and corrective actions to overcome these conditions.

4.4.1 No Interrupt (Invalid Code). Int_type = 000b

This condition occurs when the driver detects an interrupt, but ThunderLAN did not cause this interrupt. This indicates a hardware error that is caused by other hardware. The driver can be configured to ignore this interrupt and not acknowledge it. An error counter may be used for such occurrences.

4.4.2 Tx EOF Interrupt. Int_type = 001b

Tx EOF and Tx EOC interrupt handling depends on the Tx interrupt threshold used. The interrupt threshold counter is part of Texas Instruments Adaptive Performance Optimization™ (APO) algorithm. More information on APO can be found in the *ThunderLAN Adaptive Performance Utilization Technical Brief* (TI literature number SPWT089). There are two main options described below.

In the first option, the interrupt threshold is set to a nonzero number. ThunderLAN does not interrupt until it has encountered the number of Tx EOFs given to it by the Ack_Count register. In this way, the host is able to acknowledge multiple Tx EOFs in a single interrupt call. The host must count the number of frames it has transmitted by counting the frames with the Frm_Cmp bit set in the CSTAT field and must use this number in the Ack_Count field while acknowledging.

A special case of this first option is when the interrupt threshold is set to a value of 1. This gives an interrupt for each Tx EOF encountered (one frame = one Tx EOF = one interrupt). In this case, ThunderLAN interrupts the host each time it transmits a frame. The host must then acknowledge this interrupt by writing an acknowledge count of 1 to HOST_CMD with the appropriate bits set.

Depending on the application, a continuous transmit channel may not be feasible. In other words, there may not be enough frames to create a continuous transmit list, and the adapter issues a Tx EOF and a Tx EOC for every frame transmitted (one frame = one Tx EOF + one Tx EOC = two interrupts). A Tx GO command must be executed each time a frame is transmitted, as the Tx channel has been stopped by ThunderLAN (as evidenced by the Tx EOC).

This condition can be avoided by loading the interrupt threshold with a 0. Doing this disables all Tx EOFs. ThunderLAN only interrupts the host for Tx EOCs (one frame = one Tx EOC = one interrupt). This simplifies the driver, since it only has to acknowledge one interrupt per frame.

4.4.3 Statistics Overflow Interrupt. Int_type = 010b

This interrupt is given when one of the network statistics registers is halfway filled. The driver:

- ❑ Reads all the statistics registers, thereby clearing them
- ❑ Acknowledges the interrupt, then exits

When reading the statistics registers, it is a good idea to use the Adr_Inc bit in the DIO_ADR register. Using the Adr_Inc feature autoincrements the DIO address by 4 on each DIO read. This feature saves on driver code and programming time.

4.4.4 Rx EOF Interrupt. Int_type = 011b

An Rx EOF interrupt occurs when an Rx frame has been successfully received. At this time there is no Rx interrupt threshold, so each frame immediately triggers an interrupt. On receiving an Rx EOF the driver:

- ❑ Reads the Data_Address and Data_Count pointers
- ❑ Determines how many bytes to copy to the Rx buffer, or whether there is a buffer into which a frame can be copied
- ❑ Copies data into the Rx buffer pointed to by the fragments (Data_Count, Data_Address field pairs)
- ❑ Passes control of the Rx buffer to the upper layer
- ❑ Relinks the lists
- ❑ Acknowledges the interrupts, then exits

4.4.5 Dummy Interrupt. Int_type = 100b

A dummy interrupt can be created by asserting the Req_Int bit in the HOST_CMD register. This interrupt can be useful as a driver-definable interrupt, as well as in hardware diagnostics. This interrupt ensures that the adapter is open on the wire. The driver simply acknowledges this interrupt and exits.

4.4.6 Tx EOC Interrupt. Int_type = 101b

A Tx EOC interrupt occurs when ThunderLAN encounters a forward pointer of 0 in the Tx list structure or when the Ld_Thr bit is loaded with 0. In this routine the driver:

- Gets the pointer to the Tx buffer queue
- Checks the list CSTAT to see if a frame has been transmitted
  - If no, acknowledges the interrupt and exits
  - If yes, writes a 0 to CSTAT to make the list invalid

4.4.7 Network Status Interrupt. Int_type = 110b and Int_Vec = 00h

A network status interrupt occurs when a PHY status change has been detected and ThunderLAN has seen an interrupt on the MDIO line. This interrupt type occurs only if a physical interface (PHY) with enhanced media independent interface (MII) support is used.

Some other causes of a status interrupt occur when the Tx and Rx channels are stopped using a STOP command. The following shows the flow for a status interrupt routine:

- Reads the NetSts register
- Clears the NetSts register. NetSts can be easily cleared by writing what has been read back into it.
- Reads the NetMask register
- Uses NetMask to ignore the NetSts bits, which are disabled
- Evaluates the following conditions:
  - If the MIRQ bit is set, there was an MII interrupt from the PHY. For voice grade (VG) operation, this is an indication that the PHY needs to retrain.
  - If the HBEAT bit is set, a heartbeat error was detected.
  - If the TXSTOP bit is set, a Tx STOP command was given and the Tx channel is stopped.
  - If the RXSTOP bit is set, an Rx STOP command was given and the Rx channel is stopped.
- Acknowledges interrupts and exits

4.4.8 Adapter Check Interrupt. Int_type = 110b and Int_Vec ≠ 00h

An adapter check interrupt occurs when ThunderLAN enters an unrecoverable state and must be reset. This unrecoverable condition occurs when ThunderLAN does not agree with the parameters given to it by the driver or when it does not agree with the external hardware. On an adapter check, the driver:

- Disables interrupts
- Reads the adapter check code from the CH_PARM register
- Clears any Tx queued transmissions. In an adapter check, ThunderLAN is not on the network wire.
- Reads the statistics registers, since these are lost when ThunderLAN is reset
- Performs a software reset by asserting the Ad_Rst bit in the HOST_CMD register
- Exits the routine but does not acknowledge, since reset clears the interrupt

The adapter check error status is readable from the CH_PARM register location whenever an adapter check interrupt is indicated. The status includes fields to indicate the type of error and its source.

Figure 4–1. Adapter Check Interrupt Fields

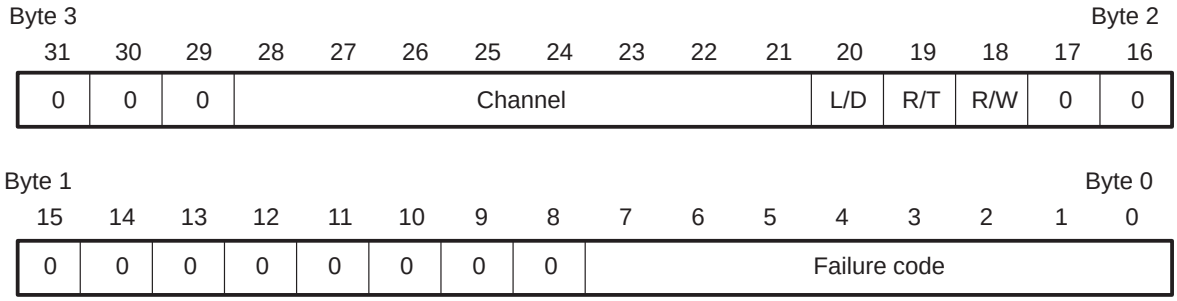


Table 4–1. Adapter Check Bit Definitions

Bit	Name	Function
28–21	Channel	This field indicates the active PCI channel at the time of the failure.
20	L/D	List not data: If this bit is set to 1, a PCI list operation was in progress at the time of the failure. If this bit is set to 0, a PCI data transfer operation was in progress at the time of the failure.
19	R/T	Receive not transmit: If this bit is set to 1, a PCI receive channel operation was in progress at the time of the failure. If this bit is set to 0, a PCI transmit channel operation was in progress at the time of the failure.
18	R/W	Read not write: If this bit is set to 1, a PCI read operation was in progress at the time of the failure. If this bit is set to 0, a PCI write operation was in progress at the time of the failure. This bit can be used to differentiate between list reads and CSTAT field writes, which are both list operations.
7–0	Failure code	This field indicates the type of failure that caused the adapter check.

Table 4–2. Adapter Check Failure Codes

Bit	Name	Function
00h		
01h	DataPar	Data parity error: Indicates that during bus master operations, ThunderLAN has detected a PCI bus data parity error, and parity error checking was enabled (the PAR_En bit in the PCI command register is set).
02h	AdrsPar	Address parity error: Indicates that ThunderLAN has detected a PCI bus address parity error, and that parity error checking is enabled (the PAR_En bit in the PCI command register is set).
03h	Mabort	Master abort: When set, indicates ThunderLAN aborted a master cycle due to a master abort. ThunderLAN master aborts a PCI data transfer if the target does not respond with a PDEVSEL# signature within six clock cycles of the start of the transfer.
04h	Tabort	Target abort: When set, indicates a ThunderLAN master cycle was aborted due to a target abort.
05h	ListErr	List error: ❑ The sum of a transmit list's data count fields was not equal to the frame length indicated in the list's frame size field. ❑ The last nonzero fragment of a receive or transmit list did not have bit 31 of the data count field set. Please note that if you are in multifragment mode and are using less than ten fragments, the fragment after the last fragment used in a receive list must have 0 in its data count field.
06h	AckErr	Acknowledge error: An attempt to overservice Rx or Tx EOF interrupts has taken place.
07h	IovErr	Int overflow error: Rx or Tx EOF interrupts have been underserviced, resulting in an overflow of the interrupt counter. The interrupt counter is ten bits wide and seven interrupt bits can be acknowledged at one time. Interrupt overflow also occurs when a Tx GO command has been given before a Tx EOC has been acknowledged. The EOC interrupt counter is only one bit wide. The same happens when an Rx GO command is given before an Rx EOC has been acknowledged.
08h–FFh		Reserved

The error status bits are only relevant for some adapter check failure codes, as indicated by the following table:

Table 4–3. Relevance of Error Status Bits for Adapter Check Failure Codes

Bit	Name	Channel	List/Data	Receive/Transmit	Read/Write
01h	DataPar	Y	Y	Y	Y
02h	AdrsPar	N	N	N	N
03h	Mabort	Y	Y	Y	Y
04h	Tabort	Y	Y	Y	Y
05h	ListErr	Y	N	Y	N
06h	AckErr	Y	EOC/EOF	Y	N
07h	IovErr	Y	EOC/EOF	Y	N

The first four adapter check codes (0x01 thru 0x04) are due to errors in the hardware. They include parity errors and PCI cycles aborted. These adapter checks reveal serious hardware errors; please verify that the attached hardware is correct.

The next three adapter check codes (0x05 through 0x07) are due to inconsistencies between ThunderLAN and the driver. These include errors in the lists where the frame size given to ThunderLAN does not match the actual frame size. They also include instances where the driver and ThunderLAN do not agree on how many EOFs to acknowledge. This is a serious mistake, since it means that frames have been lost. These adapter checks show faults in the driver-hardware interaction that must be resolved.

4.4.9 Rx EOC Interrupt. Int_type = 111b

A Rx EOC occurs when ThunderLAN encounters a forward pointer of 0 in the receive list chain. A 0 forward pointer is an indication that a receive buffer is not available, and ThunderLAN shuts off the receive process. There is a potential for frame loss if an Rx EOC occurs when the receive channel is stopped, so this condition must be avoided or the channel restarted as soon as possible using the following steps:

- ❏ Move the pointer to the top of the Rx list and find it’s physical address.
- ❏ Write it to the CH_PARM register.
- ❏ In the HOST_CMD register, acknowledge the Rx EOC and give the Rx GO command in the same 32-bit move.

Chapter 5

List Structures

ThunderLAN controllers use a list processing method to move data in and out of the host's memory. A list is a structure in host memory which is composed of pointers to data. The list contains information telling ThunderLAN where in the host memory to look for the data to be transmitted or where the receive buffer is located. This chapter discusses the advantages of using a system of linked list structures to support continuous network transmission and reception. It also discusses list format and how to effectively manage this system of linked lists by the use of interrupts.

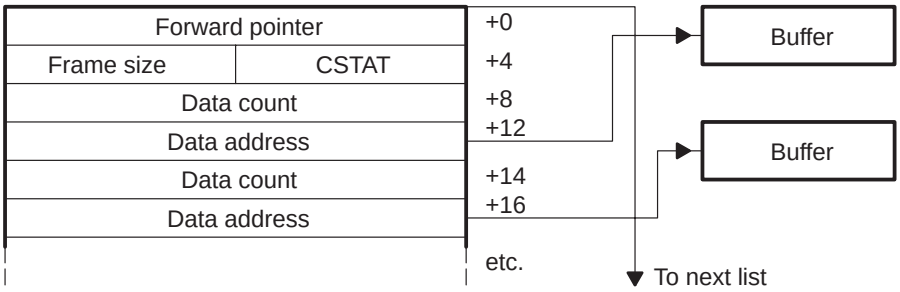
Topic **Page**

5.1 List Management

Some of the more commonly used list management terms are defined here:

List A list is a structure in host memory which is composed of pointers to data. The list includes information on the location of a frame, its size, and its transmission/receive status. A list can represent only one frame, but lists can be linked through the forward pointer. This way, multiple frames can be represented by linked lists.

Figure 5–1. List Pointers and Buffers



Buffer A buffer is an allocated area in host memory where a frame fragment is located. It is pointed to by the list structure. The buffer serves as a location where ThunderLAN DMAs a received frame and from where ThunderLAN DMAs a frame to be transmitted. A buffer can store a whole frame or part of one. It may not hold more than one frame. A list may point to one or more buffers for the frame associated with those buffers.

Frame A frame is the data that is transmitted on the network. A frame can use one or multiple buffers.

GO Command A GO command tells ThunderLAN to initiate frame transmission or reception. ThunderLAN uses the list structure to determine which buffers to use in this process.

Ack Acking is the process by which the host driver acknowledges to ThunderLAN the number of frames it has processed.

A properly written ThunderLAN driver is able to work rapidly and use the CPU infrequently, since the driver only needs to build and maintain a small list for each frame. The actual data transfer to and from the host is handled by ThunderLAN in hardware. This frees the host CPU for other applications.

Lists can also be linked by putting the forward pointer in one list at the beginning of the following list. Linking lists allows ThunderLAN to process more than one frame without having the driver issue a separate transmit/receive (Tx/Rx) open channel command (GO command) for each frame. Moreover, the driver

can keep the transmit and receive channels continuously open by freeing up buffers and relinking lists faster than frames are transferred by ThunderLAN. This is important in receive operations where the Rx channel must be open continuously to avoid losing frames from the network.

All list processing and management operations are done in host memory. The driver only needs to access ThunderLAN's internal registers when opening transmit or receive channels, when acknowledging the number of frames that it has processed, or when reading the controller statistics.

Figure 5–2. Linked List Management Technique

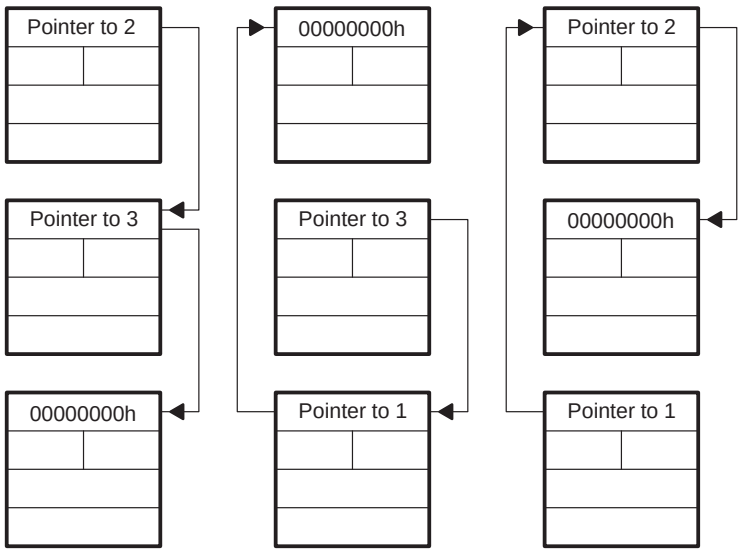


Figure 5–2 is an example of a typical three-list management technique, where the pointers are relinked sequentially. The lists are linked by pointing the forward pointer in the previous list to the address of the next list.

The first list structure is shown on the left where list 1's forward pointer points to the physical address of list 2, and list 2's forward pointer points to the physical address of list 3. List 3 has a forward pointer equal to 0x00000000h.

When ThunderLAN uses list 1, it updates the CSTAT field to show frame completion. The driver must look at the CSTAT to determine when to update the pointers. When the Frm_Cmp bit is set in the CSTAT, the driver can free up the list and the buffers. It does so by clearing the CSTAT, setting the forward pointer to 0, and writing the physical address of the forward pointer of the last list in the chain. If done quickly enough, the driver can continue to append the lists and implement a continuous transmit or receive channel. Figure 5–2 shows how the list pointers look during this appending process.

A driver is not limited in the number of lists it can manage as long as there is memory to put them in. The question then arises as to how many lists are appropriate for a certain application. The number of lists allocated should be just enough to allow the driver to use the full wire bandwidth on Tx and to handle the Rx data from the wire.

In a network client application where some data transfer may occur between the Rx buffer pool and another location in the client, the data transfer routines must be as efficient as possible. The data transfer time between the host and ThunderLAN is the copy time. This copy time must be less than the time that it takes for the network to fill up a buffer (network transmit time) plus the time it takes to service the list (service time).

Copy time < Network transmit time + Service time

Service time includes the overhead time for copying the list header and servicing the interrupts and the list. If the copy time does not meet this criterion it may be necessary to add an additional Rx list and buffer to your driver application.

An efficient driver actually takes up significantly less memory space than a less efficient driver, and it is able to use more network bandwidth and less CPU. This is because a more efficient driver uses fewer memory-consuming lists and buffers, while maintaining the same throughput. Ensuring that data transfer operations are clean and efficient helps improve the throughput and size of the driver.

A client driver can be optimized so that only one transmit list is required. Using one transmit provides good performance and simplifies the driver design. A server driver, where maximum performance is important, can be achieved with about 11 transmit lists. A client driver operates well with three receive lists. A server driver requires more to receive the network traffic. The specific number of receive and transmit lists depends on the efficiency of the driver and the machine used.

5.2 CSTAT Field Bit Requirements

Texas Instruments specifies that some bits in the CSTAT field should be set to 1, but tells you to ignore them. This is because these bits are ignored by the adapter. The ThunderLAN CSTAT is very much like that in TI380 products. Bit 12 in ThunderLAN corresponds to bit 3 in the TI380 CSTAT FRAME_END bit. Bit 13 in ThunderLAN corresponds to bit 2 in the TI380 CSTAT FRAME_START bit. We define bits 12 and 13 as 1, since that is the way that they would appear on TI380 drivers where one list handles one frame only. ThunderLAN was designed so that a TI380 driver could be easily modified to become a ThunderLAN driver.

Texas Instruments has reserved the use of these bits for future applications. Setting these bits to any other value than 1 may cause problems with later versions of ThunderLAN.

5.3 One-Fragment Mode

When the GO command is given on either transmit or receive, ThunderLAN DMAs the whole list, even though the driver only uses a limited number of fragments on that list.

In the case of a receive list, the driver has the option to force ThunderLAN to DMA a one-fragment list. This is accomplished by setting the One_Frag bit in the NetConfig register to 1. In one-fragment mode, ThunderLAN only needs to DMA a 16-byte list instead of an 88-byte list. This helps to cut down on PCI bandwidth use. Currently, there is no one-fragment mode for transmit.

5.4 Receive List Format

Figure 5–3. Receive List Format – One_Frag = 0

Byte 3	Byte 2	Byte 1	Byte 0	List offset
Forward pointer				0x00
Frame size		Receive CSTAT		0x04
Data count				0x08
Data address				0x0C
Data count				0x10
Data address				0x14
Data count				0x18
Data address				0x1C
Data count				0x20
Data address				0x24
Data count				0x28
Data address				0x2C
Data count				0x30
Data address				0x34
Data count				0x38
Data address				0x3C
Data count				0x40
Data address				0x44
Data count				0x48
Data address				0x4C
Data count				0x50
Data address				0x54

Note: All receive lists must start on an eight-byte address boundary.

Figure 5–4. Receive List Format – One_Frag = 1

Byte 3	Byte 2	Byte 1	Byte 0	List offset
Forward pointer				0x00
Frame size		Receive CSTAT		0x04
Data count				0x08
Data address				0x0C

Table 5–1. Receive Parameter List Fields

Field	Definition
Forward pointer	This full 32-bit field contains a pointer to the next receive parameter list in the chain. The three LSBs of this field are ignored, as lists must always be on an eight-byte address boundary. When the pointer is 0, the current receive parameter list is the last in the chain. The adapter processes receive parameter lists until it reads a list with a 0 forward pointer. When a valid frame has been received into the data area pointed to by this list and the receive CSTAT complete field is written, the receive channel stops. An Rx EOC interrupt is raised for the channel, but this is seen by the host (because of interrupt prioritization) until all outstanding Rx EOF interrupts have been acknowledged. The system must update the forward pointer as a single 32-bit write operation to ensure the adapter does not read it during update. The adapter does not alter this field.
Receive CSTAT	This 16-bit parameter is written to by the host when the receive parameter list is created. It is overwritten by the adapter to report frame completion status. When initially written to by the host, this field is referred to as the receive CSTAT request field. After a frame finishes receiving, the adapter overwrites this field and it is referred to as the receive CSTAT complete field. This indicates completion of frame reception, not completion of the receive command. The bit definitions for these two fields are contained in later tables.
Frame size	This 16-bit field contains the number of bytes in the received frame. This field is written by the adapter after frame reception. The frame size value does not include any frame delimiters, preambles, postscripts, or CRC fields (except in pass_CRC mode). The adapter ignores the initial value of this field.
Data count	This 32-bit field indicates the maximum number of frame data bytes to be stored at the address indicated by the following data address field. There can be up to ten data count/data address parameters in a list. A data count of 0 is necessary in the next data count field of the list if you are in multifragment mode and are using less than nine fragments.
Data address	This 32-bit field contains a pointer to a fragment (or all) of the frame data storage in host (PCI) address space. Data address is a full-byte address. Frame fragments can start (and end) on any byte boundary.

Figure 5–5. Receive CSTAT Request Fields

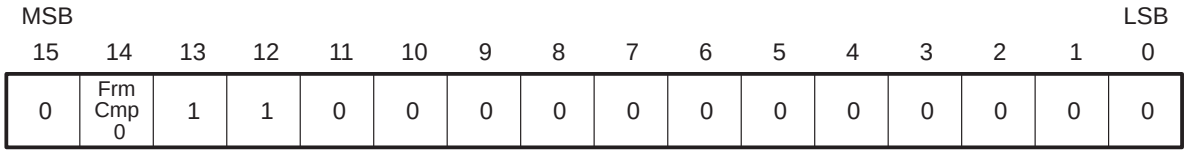


Table 5–2. Receive CSTAT Request Bits

Bit	Name	Function
15	0	Ignored by adapter. Set to 0
14	Frm_Cmp 0	Frame complete: Ignored by adapter. Set to 0. Setting the Frm_Cmp bit to 0 is good programming practice.
13	1	Ignored by adapter. Set to 1
12	1	Ignored by adapter. Set to 1
11	0	Ignored by adapter. Set to 0
10	0	Ignored by adapter. Set to 0
9	0	Ignored by adapter. Set to 0
8	0	Ignored by adapter. Set to 0
7–0	0	Ignored by adapter. Set to 0

Figure 5–6. Receive CSTAT Complete Fields

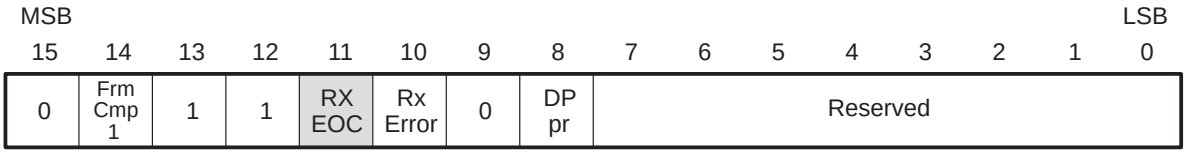
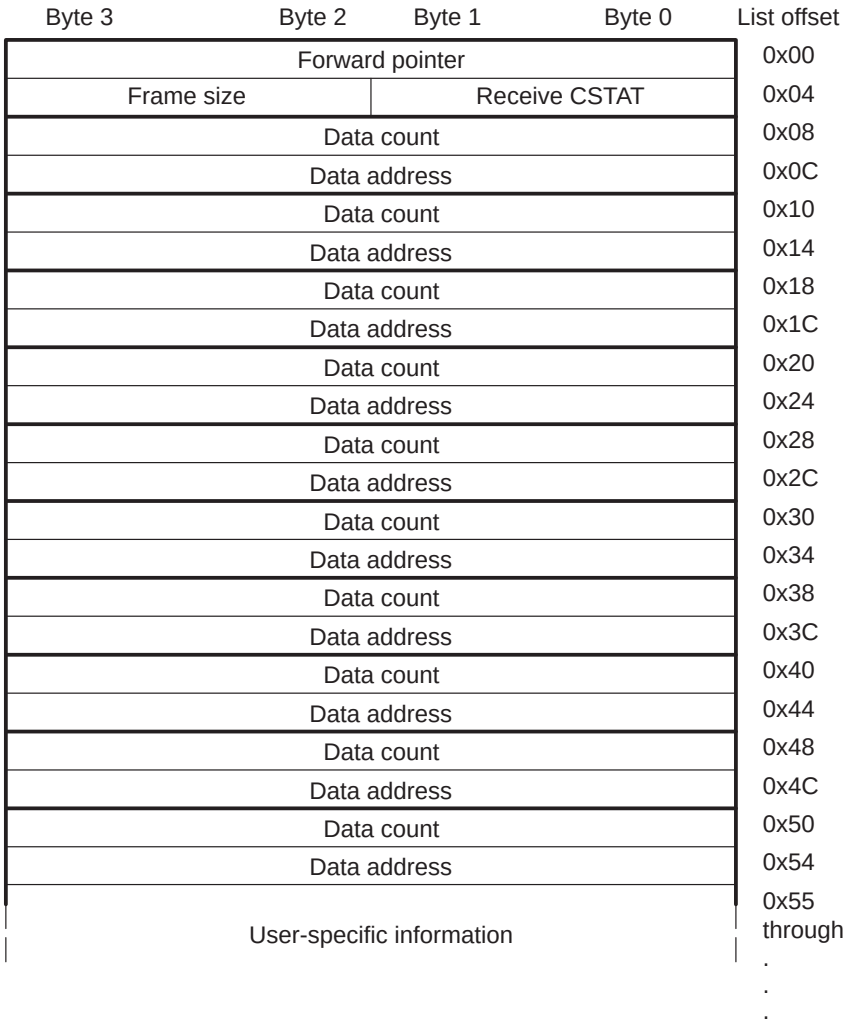


Table 5–3. Receive CSTAT Complete Bits

Bit	Name	Function
15	0	Same value as previously set by the host in CSTAT request field
14	Frm_Cmp 1	Frame complete: Set to 1 by the adapter to indicate the frame has been received
13	1	Same value as previously set by the host in CSTAT request field
12	1	Same value as previously set by the host in CSTAT request field
11	RX EOC	RX EOC: If RX EOC is disabled by the *Interrupt Disable register* no interrupts will be generated. This bit will serve as an indication of RX EOC in that case. This bit will also be set when interrupts are enabled.
10	Rx_Error	Error frame: Frames with CRC, alignment, or coding errors are passed to the host if the CAF (Copy All Frames), and PEF (Pass Error Frames) option bits are set. Such frames can be identified through the Rx_Error bit. These frames have this bit set to a 1, all good frames have this bit set to a 0.
9	0	Same value as previously set by host in the CSTAT request field
8	DP_pr	Demand priority frame priority: This bit indicates the transmission priority of received demand priority frames. A value of 0 indicates normal transmission, a value of 1 priority transmission. In CSMA/CD mode, this bit is always written as a 0.
7–0	0	Reserved

5.5 Transmit List Format

Figure 5–7. Transmit List Format



Note: All transmit lists must start on an eight-byte address boundary.

Table 5–4. Transmit Parameter List Fields

Field	Definition
Forward pointer	This 32-bit field contains a pointer to the next transmit parameter list in the chain. The three LSBs of this field are ignored, as lists must always be on an eight-byte address boundary. When the forward pointer is 0, the current transmit parameter list is the last in the chain. The adapter processes transmit parameter lists until it reads a list with a 0 forward pointer. Once the frame pointed to by this list has been read into the adapter and the transmit CSTAT complete field written, the transmit channel stops. A Tx EOC interrupt is raised for the channel, but this is not seen by the host (because of interrupt prioritization) until all outstanding Tx EOF interrupts have been acknowledged. The system must update the forward pointer as a single 32-bit write operation to ensure the adapter does not read the field while it is being updated. The forward pointer is read at the same time as the rest of the list structure. The adapter does not alter this field.
Frame size	This 16-bit field contains the total number of bytes in the transmit frame. This field must be equal to the sum of the data count fields, or an adapter check occurs.
Transmit CSTAT	This 16-bit parameter is written by the host when the transmit parameter list is created. It is overwritten by the adapter to report frame transfer status. When initially written by the host, this field is referred to as the transmit CSTAT request field. After a frame is transferred into the adapter, the adapter overwrites this field and it is referred to as the transmit CSTAT complete field. This indicates completion of frame transfer into the adapter FIFO, not frame transmission or completion of the transmit command. The bit definitions for these two fields are defined in later tables.
Data count	This 32-bit field indicates the number of frame data bytes to be found at the address indicated by the following data address field. There can be a maximum of ten data count/data address parameters in a list. If bit 31 is set to a 1, then this is the last data count in the list. A data count of 0 is only permitted when it follows the last fragment in multifragment mode. This is only necessary when nine or less fragments are used.
Data address	This 32-bit field contains a pointer to a fragment (or all) of the frame data in host (PCI) address space. Data address is a byte address. Frame fragments can start and end on any byte boundary.

Figure 5–8. Transmit CSTAT Request Fields

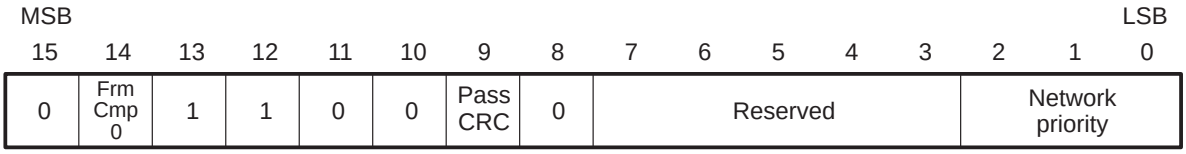


Table 5–5. Transmit CSTAT Request Bits

Bit	Name	Function
15	x	Ignored by adapter. The value in this bit is a don't care.
14	Frm_Cmp 0	Frame complete: Ignored by adapter. Should be set to 0. Setting the Frm_Cmp bit to 0 is good programming practice.
13	1	Ignored by adapter. Set to 1
12	1	Ignored by adapter. Set to 1
11	0	Ignored by adapter. Set to 0
10	0	Ignored by adapter. Set to 0
9	Pass_CRC	Pass CRC: When this bit is set to a 1, the adapter uses the last four bytes of frame data as the frames CRC. The integrity of this CRC is not checked and it is handled just like the data payload. If this bit is set to a 0, CRC is generated by the adapter as normal.
8	0	Ignored by adapter. Should be set to 0
7–3	Reserved	This field is Ignored by the adapter, but should be set to 0.
2–0	Network priority	Network priority request: Indicates the network transmission priority to be used for the frame when network protocol types 2 or 3 are selected. The value in this field is passed to the external protocol engine across the 802.3u MII on the MTXD[3::1] signal lines during transmission requests. This field has no meaning if network protocol types other than 2 or 3 are selected.

Figure 5–9. Transmit CSTAT Complete Fields

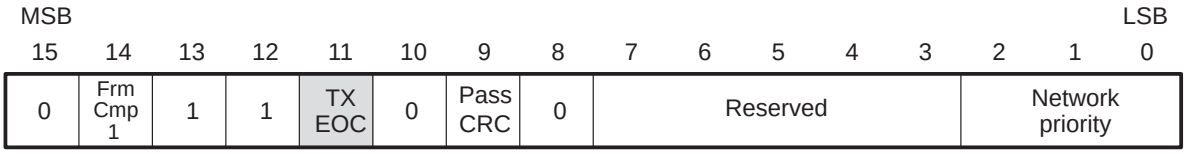


Table 5–6. Transmit CSTAT Complete Bits

Bit	Name	Function
15	x	Same value as previously set by the host in the CSTAT request field
14	Frm_Cmp 1	Frame complete: Set to 1 by the adapter to indicate the frame has been transmitted into the controller's internal FIFO
13	1	Same value as previously set by the host in CSTAT request field
12	1	Same value as previously set by the host in CSTAT request field
11	TX EOC	TX EOC: If TX EOC is disabled by the *Interrupt Disable register* no interrupts will be generated. This bit will serve as an indication of TX EOC in that case. This bit will also be set when interrupts are enabled.
10	0	Same value as previously set by the host in CSTAT request field
9	Pass_CRC	Same value as previously set by the host in CSTAT request field
8	0	Same value as previously set by the host in CSTAT request field
7–3	0	This field is ignored by the adapter, but should be set to 0.
2–0	Network priority	Same value as previously set by the host in CSTAT request field

Chapter 6

Transmitting and Receiving Frames

This chapter describes the structure and format for transmitting and receiving frames using ThunderLAN. Frames are units of data that are transmitted on a network. These must appear in a consistent, logical format to be recognized. The chapter also describes the method you must use to create a linked list structure, which is necessary to start frame reception and transmission.

Topic **Page**

6.1 Frame Format

The following describes the configuration of the data units which ThunderLAN transmits and receives. ThunderLAN looks for this format to create the linked structures it uses in transmitting and receiving data (see subsection 6.2, GO Command).

6.1.1 Receive (Rx) Frame Format

The adapter receive channels are used to receive frames from other nodes on the network. The ThunderLAN adapter allows received frame data to be fragmented into up to ten pieces. However, the adapter provides the data in a consistent, logical format as shown below. This logical format is different for token ring and Ethernet frame formats.

Figure 6–1. Token Ring Logical Frame Format (Rx)

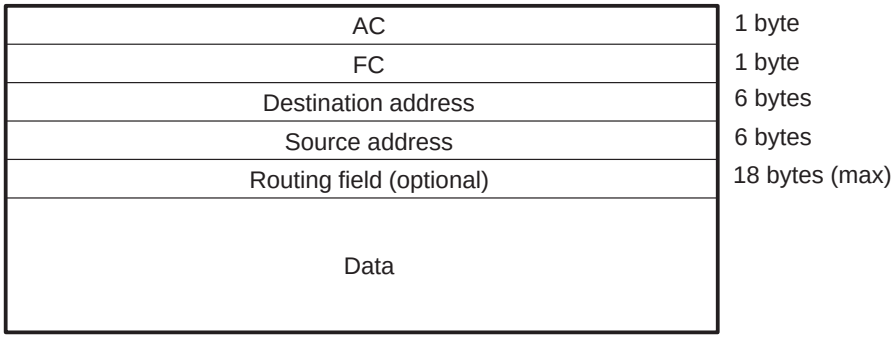
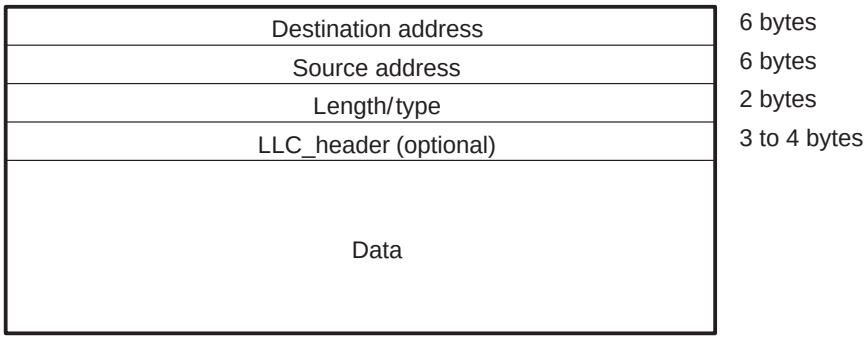


Figure 6–2. Ethernet Logical Frame Format (Rx)



6.1.2 Transmit (Tx) Frame Format

The adapter transmit channels are used to transmit frames to other nodes on the network. The ThunderLAN adapter allows transmitted frame data to be fragmented into up to ten pieces. However, the adapter expects the concatenation of these fragments to be in a consistent, logical format as shown below. This logical format is different for token ring and Ethernet frame formats.

Figure 6–3. Token Ring Logical Frame Format (Tx)

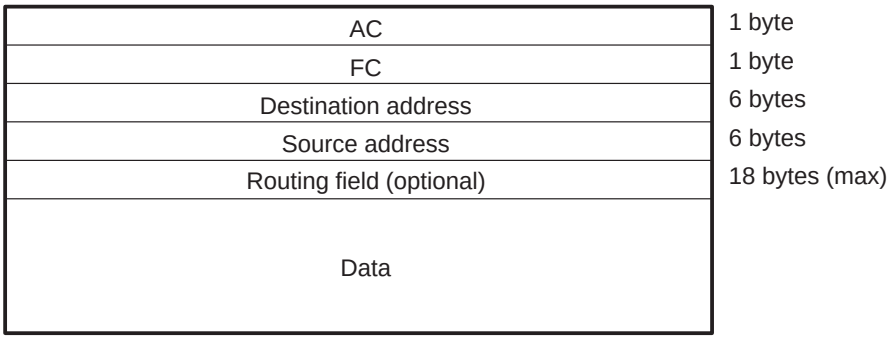
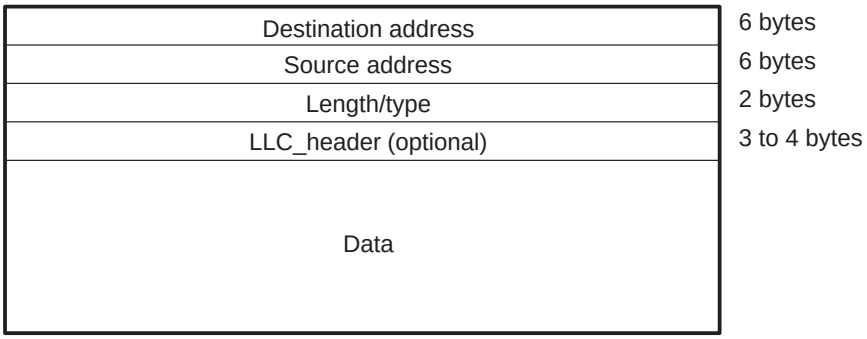


Figure 6–4. Ethernet Logical Frame Format (Tx)



6.2 GO Command

To transmit and receive data, the ThunderLAN driver must create a linked list of frames. This subsection describes the steps to create such a linked list, and the process for initiating transfer by using a GO command.

6.2.1 Starting Frame Reception (Rx GO Command)

To create a linked receive list structure the driver:

1) Allocates memory for the list structures and receive buffers

2) Aligns the list to the nearest octet (byte) boundary

3) Links the lists as follows:

   a) Determines the physical address of the next receive list and writes it to this list's forward pointer. This links the two lists together.

   b) Initializes the CSTAT field for this list by setting bits 12 and 13 to 1.

   c) Determines frame size for the Rx buffers (generally the maximum allowable frame size) and writes it to the frame size field.

   d) Sets the data count to the length of the receive buffer. This is a flag that tells ThunderLAN that it is the last fragment.

   e) Calculates the list's receive buffer's physical address and writes it to the data address.

4) Sets up the next list

After all lists are complete, ThunderLAN goes to the last list's forward pointer and sets it to 0x00000000h. To begin frame reception the host:

1) Creates a linked list of receive lists pointing to the receive buffers

2) Writes the address of the first list to CH_PARM

3) Brings ThunderLAN out of reset, if necessary, by writing 0x0000h to the least significant word (LSW) of HOST_CMD

4) Writes a 1 to the GO bit of HOST_CMD with the receive channel selected

This assumes the interrupt pacing timer has been initialized. If not, write to HOST_CMD with the Ld_Tmr bit set and the pacing timer time-out value in the Ack_Count field.

For frame reception, the driver must allocate enough memory for the receive lists and buffers. The lists must be octet aligned. They are linked by having the

forward pointer point to the next available list. The last list should have a forward pointer of 0. You must then initialize the CSTAT fields in the lists.

Opening a receive channel works in much the same way as opening a transmit channel. You must first write the address of the beginning of the list chain to CH_PARM and then give the receive open channel (Rx GO) command. ThunderLAN DMAs the data from its internal FIFO to the receive buffer pointed to in the list structure.

ThunderLAN writes the data count field with the data size when the frame is complete. The Frm_Cmp bit in the receive CSTAT complete field is set when the receive data is completely DMAed into the receive buffer. ThunderLAN then gives the host an Rx EOF interrupt. The driver looks at the Frm_Cmp bit to determine when the receive frame is complete and acknowledges the interrupt. The driver frees up the buffer for a new frame.

The driver makes the old list's forward pointer equal to 0. The previous list's forward pointer can be set up to point to this list. If done quickly, ThunderLAN always has a receive buffer to place frames in and the receive channel remains open. If ThunderLAN encounters the 0 forward pointer (meaning that it has filled the last buffer available to it), it gives the host an Rx EOC interrupt and stops the receive channel.

Implementing continuous receive reduces the possibility of losing frames due to not having receive buffers. In continuous receive, the driver acknowledges Rx EOF interrupts as each frame is delivered to the host. The driver manages the receive lists and buffers so that ThunderLAN always has a free buffer to place data in.

In case of an Rx EOC interrupt, the driver frees up buffers for the receive process, creates a system of linked lists, and restarts the receive channel by issuing an Rx GO command.

The Rx GO command is used to pass a receive list structure to ThunderLAN. ThunderLAN uses the list structure to DMA the received data from the network. There are two steps involved in issuing an Rx GO command:

- In the CH_PARM register, write the physical address of the beginning of the list structure. ThunderLAN uses this physical address to DMA data to host memory.

- In the HOST_CMD register, write the appropriate bits to start the channel. The Rx GO command writes to the GO bit and selects the transmit channel by writing to the Ch_Sel field. Set R/T to 1, indicating a receive command.

The HOST_CMD register can be written in a single, 32-bit operation. This implies that several commands can be combined in one operation. An Rx EOC interrupt can be acknowledged and Rx GO commands can be reissued in a single operation.

6.2.2 Starting Frame Transmission (Tx GO Command)

To create a linked transmit list structure the driver:

1) Allocates enough memory for the list structures and transmit buffers
2) Aligns the list to the nearest octet (byte) boundary. This ensures that the PCI bus does not insert any wait cycles to align the list with its internal 64-bit architecture.
3) Starts linking lists as follows:
    a) Determines the physical address of the next transmit list and writes it to this list's forward pointer. This links the two lists together.
    b) Initializes the CSTAT field for this list by setting bits 12 and 13 to 1
    c) Initializes the frame_size to the transmit frame's size
    d) Sets the data count to 0x80000000h. This is a flag that tells ThunderLAN that it is the last fragment. ThunderLAN, however, DMAs the entire ten fragments even though it may only use one fragment.
    e) Calculates the list's transmit buffer's physical address and writes it to the data address
4) Sets the next transmit list

After all lists are complete, go to the last list's forward pointer and set it to 0x00000000h. To begin frame transmission, the host:

1) Creates a linked list of transmit lists pointing to the transmit frames
2) Gets the addresses of the next available Tx list
3) Clears the CSTAT field in the list
4) Ensures that the frame is padded to 60 bytes for a minimum IEEE 802.3 standard frame length of 64 bytes (60 bytes + 4-byte CRC)
5) Copies the media header to the Tx list transmit data buffer
6) Moves the frame length and the data buffer pointer to the transmit list
7) Disables interrupts

8) Gives the TX GO command by writing the address of the first available list to the CH_PARM register

9) Writes a 1 to the GO bit of the HOST_CMD register, with the transmit channel selected

This assumes the transmit interrupt threshold has been initialized. If not, write to HOST_CMD with the Ld_Thr bit set and the threshold value in the Ack_Count field.

For frame transmission, the driver first allocates memory for the transmit lists and buffers and octet (byte) aligns the transmit lists. Next, the driver links the list by having the forward pointer point to next available list. The forward pointer must be 0 in the last list used. Next, the driver initialize the CSTAT fields in the lists.

After this system of linked lists is created, the driver writes the address of the first list to the CH_PARM register. This tells ThunderLAN where the first linked list is. After this, a Tx GO command is given. ThunderLAN initiates the DMA of data into its internal FIFO and then ThunderLAN transfers at least 64-bytes, or the value given in the Acommit register, into its internal FIFO before starting frame transmission.

When ThunderLAN finishes transferring data into its internal FIFO, it sets the Frm_Cmp bit in the CSTAT field. (ThunderLAN sets the bit when the frame is transmitted to the FIFO, not when network transmission is complete). The driver looks into the Frm_Cmp bit to verify frame transfer. Depending on the value loaded into the Ld_Thr bit in HOST_CMD, a Tx EOF interrupt is given to the host. The driver then needs to acknowledge the number of frames it has sent to ThunderLAN. This is important, as ThunderLAN and the driver have to agree on the number of frames sent. If there is a discrepancy, and the host acknowledges more frames than ThunderLAN has sent, ThunderLAN assumes that a serious hardware error has occurred and an adapter check 06h AckErr will follow.

When the driver determines the Frm_Cmp bit is set, it frees up the list and buffer for the next transmit list. When the driver wishes to transmit a frame, it sets up the Tx list to point to the buffer that holds the frame. It writes a forward pointer of 0 to make this the last list in the chain. Finally, the previous list's forward pointer must be changed to write to the beginning of this list. ThunderLAN then transmits this frame as it goes down the linked lists. There are situations when ThunderLAN has completed transmitting the previous list and has seen the 0 forward pointer, in which case it closes the Tx channel and gives a Tx EOC interrupt. The driver then acknowledges the EOC and resumes transmitting by issuing another Tx GO command.

Depending on the value loaded into the Ld_Thr bit in the HOST_CMD register, ThunderLAN gives a Tx EOF interrupt after processing the number of frames specified. In this case, the driver acknowledges the number of frames that it has processed. Again, the driver has to look into the Frm_Cmp bit of the CSTAT field to determine the number of frames that have been processed.

The transmit open channel command (Tx GO) initializes frame transmission into ThunderLAN's internal transmit FIFO and subsequently to the network. There are two steps involved in issuing a Tx GO command:

1) In the CH_PARM register, write the physical address of the beginning of the list structure. ThunderLAN uses this physical address to DMA data from host memory.

2) In the HOST_CMD register, write the appropriate bits to start the channel. The Tx GO command writes to the GO bit, and selects the transmit channel by writing to Ch_Sel. R/T must be set to 0, indicating a transmit command.

The HOST_CMD register can be written in a single, 32-bit operation. This implies that several commands can be combined in one operation. Ack and Tx GO commands can be issued in a single operation.

Chapter 7

Physical Interface (PHY)

This chapter describes ThunderLAN support for all IEEE 802.3-compliant devices through its media independent interface (MII). These include the internal 10Base-T physical interface (PHY) and any MII-compliant networking PHYs. It also discusses IEEE 802.12-compliant devices which are supported when ThunderLAN is used in conjunction with Texas Instruments TNETE211 100VG-AnyLAN physical media independent (PMI) device. The TNETE211 implements 802.12 media access controller (MAC) state machines for 100VG-AnyLAN operation, and provides an 802.12-compatible MII.

In addition, interrupt-driven PHYs can be implemented through the use of the enhanced MII. ThunderLAN can use nonmanaged (no serial management interface) MII devices. ThunderLAN also supports bit-level PHYs.

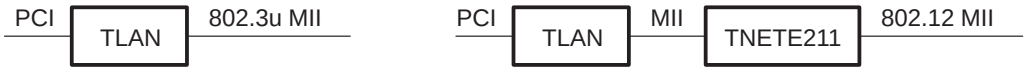
Topic **Page**

7.1 MII-Enhanced Interrupt Event Feature

ThunderLAN can connect to an external PMI device through its industry standard MII interface. A full description of the MII can be found in the 802.3u standard. The ThunderLAN MII is enhanced in two ways:

- The ThunderLAN MII can be shifted through software into a mode that supports a connection to the TNETE211. The TNETE211 device provides full 802.12 functionality and a 802.12 MII, which allows connection to a PHY for 100VG-AnyLAN. The MII mode can be selected using the MAC_Select field in the NetConfig register.

Figure 7–1. 100VG-AnyLAN Support Through ThunderLAN's Enhanced 802.3u MII



- ThunderLAN's MII passes interrupts from the attached PHY to the controller. This enables the PHYs to interrupt the host only when needed.

ThunderLAN implements the 19-signal MII shown in Table 7–1:

Table 7–1. ThunderLAN MII Pins (100M-bps CSMA/CD)

Name	Type	Function
MTCLK	In	Transmit clock: Transmit clock source from the attached PHY device
MTXD0 MTXD1 MTXD2 MTXD3	Out	Transmit data: Nibble transmit data from ThunderLAN. When MTXEN is asserted, these pins carry transmit data. Data on these pins is always synchronous with MTCLK.
MTXEN	Out	Transmit enable: Indicates valid transmit data on MTXD[3::0]
MTXER	Out	Transmit error: Allows coding errors to be propagated across the MII
MCOL	In	Collision sense: Indicates a network collision
MCRS	In	Carrier sense: Indicates a frame carrier signal is being received.
MRCLK	In	Receive clock: Receive clock source from the attached PHY
MRXD0 MRXD1 MRXD2 MRXD3	In	Receive data: Nibble receive data from the PHY. Data on these pins is always synchronous to MRCLK.
MRXDV	In	Receive data valid: Indicates data on MRXD[3::0] is valid
MRXER	In	Receive error: Indicates reception of a coding error on received data
MDCLK	Out	Management data clock: Serial management interface to PHY chip
MDIO	I/O	Management data I/O: Serial management interface to PHY chip
MRST#	Out	MII reset: Reset signal to the PHY front end (active low)

Communication with these devices is via the two MII pins MDIO and MDCLK. The MDCLK signal is sourced from the host and is used to latch the MDIO pin on the rising edge.

An MII frame consists of 32 bits, as shown in Figure 7–2 and Figure 7–3:

Figure 7–2. MII Frame Format: Read

Start delimiter	Operation code	PHY address	Register address	Turn-around	Data
01	10	AAAAA	RRRRR	Z0	DDDD DDDD DDDD DDDD

Figure 7–3. MII Frame Format: Write

Start delimiter	Operation code	PHY address	Register address	Turn-around	Data
01	01	AAAAA	RRRRR	10	DDDD DDDD DDDD DDDD

The clock cycle at the end of a transaction is used to disable the PMI from driving the MDIO pin after a register read (the quiescent cycle). ThunderLAN supports an extra feature on the serial management interface whereby the PHY can interrupt the host. The interrupt is signaled to the host on the MDIO pin one clock cycle later, during the half of the MDCLK cycle which is high.

MII-managed devices use this serial interface to access the internal register space as defined in the 802.3 or 802.12. A driver recognizes these devices by successfully reading the register space. A specific PHY can be found by reading the PHY identifier registers (locations 0x02h and 0x03h) and matching them to a known code.

For Texas Instruments PHYs and PMIs, these codes are shown below, where the xx denotes a revision:

- 0x4000501xx for the internal 10Base-T PHY
- 0x4000502xx for the TNETE211 100VG-AnyLAN PMI

One performance enhancement that ThunderLAN architecture supports is the ability to be interrupted by the PHY. This is accomplished through ThunderLAN's enhanced MII. The MII-enhanced interface allows the PHY to interrupt the host system to indicate that the PHY needs some type of service, rather than requiring the host to constantly poll the MII registers.

The interrupt mechanism described here is an extension of the 802.3u standard and does not affect MII compatibility with the existing 802.3u standard.

Servicing an interrupt typically requires the host to read the PHY generic status register followed by a read of the PHY specific status register. According to the 802.3u, the PHY-specific register is a user-defined register and is normally located between the MII registers 0x10 and 0x1f. Texas Instruments has chosen 0x12 as the location for the TLPHY_sts register.

The MII interrupt bit of this register, MINT, is bit 15, the MSB. This bit indicates that an interrupt is pending or has been cleared by the current read. The PHY-specific control register is located at 0x11 and contains two bits of importance, TINT (test interrupt) and INTEN (interrupt enable). The TINT bit is used to test the interrupt function; setting this bit forces the PHY to generate an interrupt. Clearing this bit disables this function. The function of the TINT bit is to test the

PHY interrupt function. The INTEN bit is used to enable and disable the PHY interrupt function. Setting the INTEN bit enables the PHY internal event system to generate interrupts; clearing the INTEN bit disables the PHY from generating interrupts. Interrupts from the PHY are usually generated upon a change in status that requires an interrupt indication.

Typical interrupt-generating events are shown in Table 7–2:

Table 7–2. Possible Sources of MII Event Interrupts

Name	Function
Link	Interrupt generated when the last read value is different from the current state of the link, or different from the latched link fail value if the link went OK-FAIL-OK with no register read in between.
TxJabber	Interrupt generated when the JABBER bit is changed from a 0 to a 1.
Remote fault	Interrupt generated when the remote fault (RFLT) bit is changed from a 0 to a 1.
Autonegotiation complete	Interrupt generated when the autonegotiation complete (AUTOCMPLT) bit is changed from a 0 to a 1.
Other event	Other events that may require the management information base (MIB) statistics updated if a counter has reached its half-way point. These counters should clear when read. This action also disables the counter event function.

The design of the interrupt-generating logic in the PHY should minimize the number of interrupts generated so that overall system performance is not impacted. To achieve this, the events that cause an interrupt are detected by change and not by absolute value. The host, when interrupted, reads the PHY generic status register, followed by the PHY specific status register. This clears the interrupt, as well as clearing the bits that needs to be counted in the MIB.

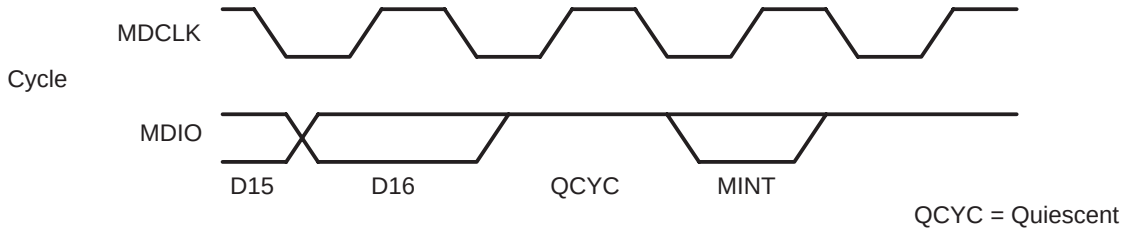
There is only one deviation from this behavior associated with the LINK bit. This bit needs to inform the host of a change in link status if the last read value was different than the current value. With the exception of LINK = FALSE during a LINK = TRUE condition, this condition is latched by the PHY and held until the host reads the link fail condition. Since the TxJabber, RemoteFault, and XtalOk conditions are either counted or start a host recovery process, they need only cause an event when they are first set. The JABBER bit should be cleared on the read that reads this bit as a 1, otherwise the next interrupt will count this set bit, corrupting the MIB statistics.

The 802.3u standard provides for communication with the PMI via the two management interface MII pins: MDIO and MDCLK. The MDCLK signal is

generated under host software control and is used to latch the MDIO pin on the rising edge.

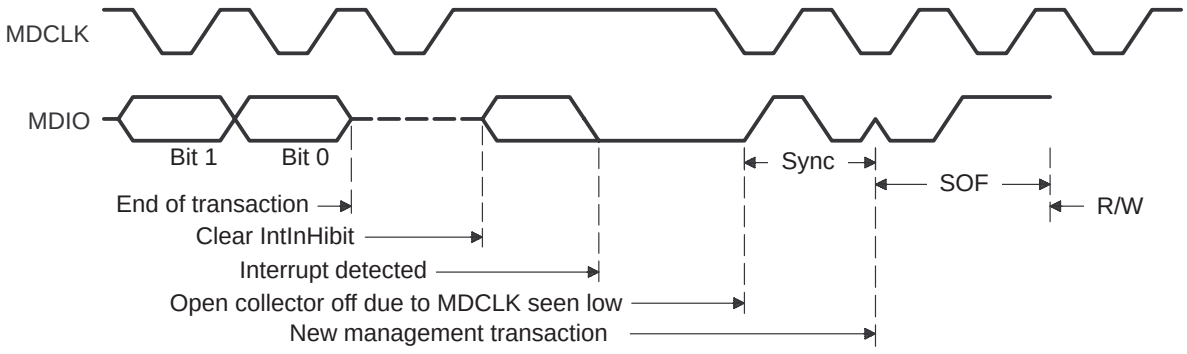
The ThunderLAN architecture expands the use of these two pins to allow the attached PHY to interrupt the host using ThunderLAN. The clock cycle at the end of a transaction on the MDIO signal is used to disable the PMI from driving MDIO after a register read (the quiescent cycle). The interrupt is signaled to the host on MDIO one clock cycle after this, for the half of the cycle when MDCLK is high.

Figure 7–4. Assertion of Interrupt Waveform on the MDIO Line



Since the Interrupt from the PHY is an open drain function, the PMI drives the MDIO low prior to the falling edge that starts the start of frame (SOF) portion of the management interface frame. During sync cycles the PHY releases the interrupt on the MDIO to allow the management entity to pull the MDIO high so a sync cycle is seen. In the diagram below, only one sync cycle is displayed, but all 32 bits of the sync cycle are the same. On the ThunderLAN side of the MII, a pullup is used to pull the MDIO signal high (no interrupt pending). The value is such that the rising edge is less than 200 ns.

Figure 7–5. Waveform Showing Interrupt Between MII Frames



7.2 Nonmanaged MII Devices

Nonmanaged MII devices do not have a management interface (MDIO and MDCLK). As such, they do not have any registers. The driver must have a keyword that denotes that the PHY used is nonmanaged.

7.3 Bit-Rate Devices

ThunderLAN supports bit-rate devices by asserting the BITrate bit in the NetConfig register. The MII is then converted into an Ethernet serial network interface (SNI). The pin conversion for this mode is:

- MRXD0 → RXD (receive data)
- MRCLK → RXC (receive clock)
- MRCLK → RXC (receive clock)
- MCRS → CRS (carrier sense)
- MTCLK → TXC (transmit clock)
- MTXD0 → TXD (transmit data)
- MTXEN → TXE (transmit enable)
- MCOL → COL (collision detect)

MTXD3, MTXD2, MTXD1, and MTXER are driven with the values in the low nibble (bits 0–3) of the Acommit register. These signals can be used to drive PHY option pins, such as loopback or unshielded twisted pair/attachment unit interface (UTP/AUI) select.

7.4 PHY Initialization

The driver initializes each MII-attached PHY. Since there may be more than one PHY attached to the MII, proper initialization ensures that one and only one PHY is active and driving the MII. (The condition where more than one PHY drives the MII at the same time is termed contention.)

Each MII-equipped PHY device has a control register at offset 0x00h. In ThunderLAN, this register is called GEN_ctl. The effect of asserting the isolate bit in this register is that the PHY does not drive any wire, port, or magnetics it is connected to, and it Hi-Zs the interface to the MII data bus. The PHY still pays attention and responds to requests from the MII management interface, consisting of the MDIO/MDCLK lines.

Nominal treatment of the attached PHY requires 32 clock cycles be applied to the PHY to synchronize the serial management interfaces. This is required on each access, unless it responds to a fast start delimiter. If a PHY has this feature, synchronization is only required on the first access. The clock cycles are generated through the NetSio register.

Appendix A

Register Definitions

This appendix contains register definitions for the TNETE100A, TNETE110A, and TNETE211 ThunderLAN implementations. ThunderLAN uses these registers to store information on its internal status and its communication with the host. This appendix describes the purpose and function of each register and provides bitmaps and descriptions of individual bits.

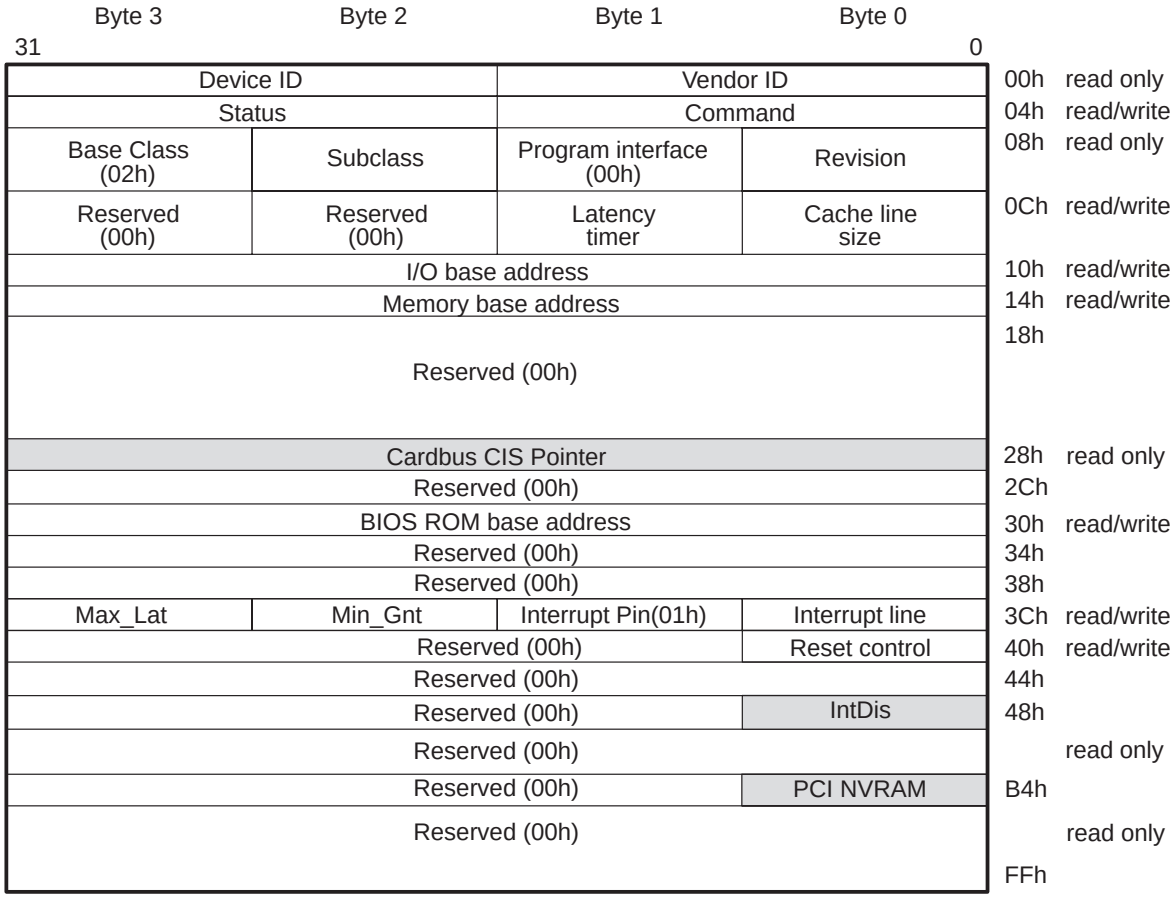
Topic **Page**

A.1 PCI Configuration Registers

The PCI specification requires all PCI devices to support a configuration register space to allow jumperless autoconfiguration. The configuration space is 256 bytes in length, of which the first 64-byte header region is explicitly defined by the PCI standard. Registers in this address space are accessed by a combination of signals. The IDSEL pin acts as a classical chip-select signal, indicating there are configuration accesses to this device. Special configuration read or write cycles are used for the accesses and individual registers are addressed using AD[7::2] and PCI bus-byte enables.

The adapter only implements mandatory or applicable configuration registers in the standard PCI header region. No adapter-specific configuration registers are defined. The adapter's PCI configuration register address map is shown in the diagram below. All reserved registers are read as 0. The registers shown shaded in the diagram can be autoloaded from an external serial EEPROM.

Figure A–1. PCI Configuration Register Address Map



A.1.1 PCI Autoconfiguration from External 24C02 Serial EEPROM

ThunderLAN allows some of the PCI configuration space registers to be loaded from an external serial EEPROM. These registers contain fixed vendor and device information. Autoconfiguration allows builders of ThunderLAN systems to customize the contents of these registers to identify their own systems, rather than using the Texas Instruments defaults.

The state of the EDIO pin during PCI reset (PRST#), enables (high) or disables (low) autoconfiguration. In order to use a 24C02 EEPROM, the EDIO line requires an external pullup. ThunderLAN enables autoconfiguration if it detects this pullup (EDIO high) during PCI reset. If autoconfiguration is not required or no EEPROM is present, the EDIO pin must be tied to ground.

The first bit written to or read from the EEPROM is the most significant bit of the byte, such as data(7). Therefore, writing the address C0h is accomplished by writing a 1 and six 0s.

ThunderLAN expects data to be stored in the EEPROM in a specific format. Nine bytes in the EEPROM are reserved for use by the adapter, starting with C0h, as shown below. The contents of the remaining 247 bytes are undefined. The EEPROM can also be read from or written to by driver software through the NetSio register. The shaded registers in Figure A–1 can be autoloaded from an external serial EEPROM.

Figure A–2. Configuration EEPROM Data Format

	Address
Vendor ID LSByte	C0h
Vendor ID MSByte	C1h
Device ID LSByte	C2h
Device ID MSByte	C3h
Revision	C4h
Subclass	C5h
Min_Gnt	C6h
Max_Lat	C7h
Checksum	C8h

Any accesses to the adapter's configuration space during autoload are rejected with a target retry. The checksum byte is an 8-bit cumulative XOR of the eight shaded bytes, starting with an initial value of AAh. The adapter uses this checksum to validate the EEPROM data. If the checksum fails, the configuration registers are set to their default (hardwired) values instead.

```
Checksum = AAh
         XOR Data(0) XOR Data(1) XOR Data(2) XOR Data(3)
         XOR Data(4) XOR Data(5) XOR Data(6) XOR Data(7);
Where XOR is a bitwise exclusive OR of the bytes.
```

A.1.2 PCI Vendor ID Register (@ 00h) Default = 104Ch

This register holds the adapter vendor ID. This register is loaded from an external serial EEPROM on the falling edge of PCI reset, during autoconfiguration. Should autoconfiguration fail (bad checksum), then this register is loaded with the TI vendor ID of 104Ch instead.

A.1.3 PCI Device ID Register (@ 02h) Default = 0500h

This register holds the adapter device ID. The register is loaded from an external serial EEPROM on the falling edge of PCI reset, during autoconfiguration.

Should autoconfiguration fail (bad checksum), this register is loaded with the ThunderLAN device ID of 0500h.

A.1.4 PCI Command Register (@ 04h)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved							SER En	Res	Par En	Reserved			BM En	Mem En	I/O En

Table A–1. PCI Command Register Bits

Bit	Name	Function
15–9	Reserved	Writes to these bits are ignored; bits are always read as 0.
8	SER_En	PSERR# driver enable: A value of 1 enables the adapter PSERR# driver. A value of 0 disables the driver.
7	Reserved	Writes to this bit are ignored; bit is always read as 0.
6	PAR_En	Parity enable: Enables the adapter PCI parity checking. A value of 1 allows the adapter to check PCI parity, a value of 0 causes PCI parity errors to be ignored.
5–3	Reserved	Writes to these bits are ignored; bits are always read as 0.
2	BM_En	Bus master enable: Enables the adapter as a PCI bus master. A value of 1 enables bus master operations, a value of 0 disables them.
1	Mem_En	Memory enable: Enables memory-mapped accesses to the adapter. A value of 0 disables response to memory-mapped accesses. A value of 1 enables response to memory-mapped accesses.
0	I/O_En	I/O Enable: Enables I/O mapped accesses to the adapter. A value of 0 disables response to I/O accesses. A value of 1 enables response to I/O accesses.

A.1.5 PCI Status Register (@ 06h)

15	14	13	12	11	10 9	8	7	6 5 4 3 2 1 0
DP err	SS err	RM ab	RT ab	Res	DEVSEL (01b)	DP det	FBB cap	Reserved

Table A–2. PCI Status Register Bits

Bit	Name	Function
15	DP_err	Detected parity error: Indicates that the adapter has detected a parity error. This bit is set even if the parity error response bit is not set. This bit can only be set by the adapter, and only cleared by the host's writing a 1 to this bit position.
14	SS_err	Signaled system error: When set, indicates ThunderLAN has asserted PSERR# due to an adapter failure. This bit can only be set by the adapter and only be cleared by the host's writing a 1 to this bit position.
13	RM_ab	Received master abort: When set, indicates ThunderLAN aborted a master cycle with a master abort. This bit can only be set by the adapter and only be cleared by the host's writing a 1 to this bit position.
12	RT_ab	Received target abort: When set, indicates a ThunderLAN master cycle was aborted due to a target abort. This bit can only be set by the adapter, and only be cleared by the host's writing a 1 to this bit position.
11	Reserved	
10 – 9	Devsel	PDevSel timing: These bits indicate the PDEVSEL# timing supported by ThunderLAN. These bits are hardwired to 01b, indicating medium speed (2 cycles) decoding of PDEVSEL#.
8	DP_det	Data parity detect*:* When set, indicates PPERR# is asserted for a cycle for which the adapter was the bus master and the parity error response bit is set. (This bit is set if the target signals PPERR# on an adapter master cycle.) This bit can only be set by the adapter and can only be cleared by the host's writing a 1 to this bit position.
7	FBB_cap	Fast back-to-back capable: This bit is hardwired to a 1 to indicate that ThunderLAN can receive fast back-to-back cycles to different agents.
6 – 0	Reserved	Writes to these bits are ignored, bits are always read as zero.

A.1.6 PCI Base Class Register (@ 0Bh)

This register is hardwired with the network controller code of 0x02h.

A.1.7 PCI Subclass Register (@ 0Ah)

This register holds the adapter PCI subclass. This register is loaded from an external serial EEPROM on the falling edge of PCI reset, during autoconfiguration. Should autoconfiguration fail (bad checksum), then this register is loaded with the other network controller code of 0x80h (there are no codes allocated to multiprotocol adapters).

A.1.8 PCI Program Interface Register (@ 09h)

This register is hardwired to 0 (no defined interface).

A.1.9 PCI Revision Register (@ 08h)

This register holds the adapter's revision. This register is loaded from an external serial EEPROM on the falling edge of PCI reset, during autoconfiguration. Should autoconfiguration fail (bad checksum), this register is loaded with 0x20h to indicate pattern generation 2.0 (PG 2.0) ThunderLAN.

A.1.10 PCI Cache Line Size Register (@ 0Ch)

This register informs the adapter of the memory system cache line size. This is used to determine the type of memory command used by ThunderLAN in PCI bus master reads.

- Memory read line is used for data reads of less than four cache lines.
- Memory read multiple is used for data reads of four or more cache lines.

A cache line size of 0 (default register state after reset) is interpreted as a cache line size of 16 bytes. Cache line sizes must be powers of 2 (0, 1, 2, 4, 8, 6, 32, 64, 128); values that are not powers of 2 are rounded down to the nearest power of 2. This register is loaded with 0 at reset.

A.1.11 PCI Latency Timer Register (@ 0Dh)

This register specifies in units of PCI bus clock cycles, the value of the adapter latency timer. This register is loaded with 0 at reset.

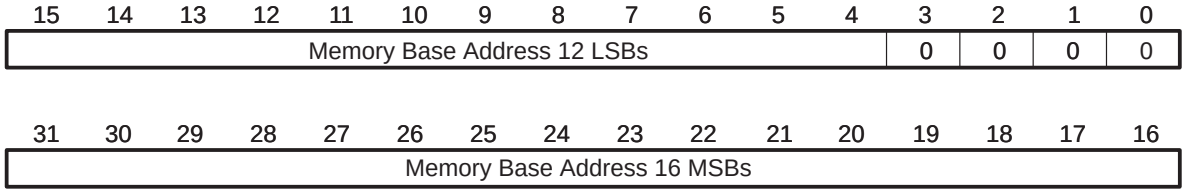
A.1.12 PCI I/O Base Address Register (@ 10h)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
I/O base address 12 LSBs												0	0	0	1

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
I/O base address 16 MSBs															

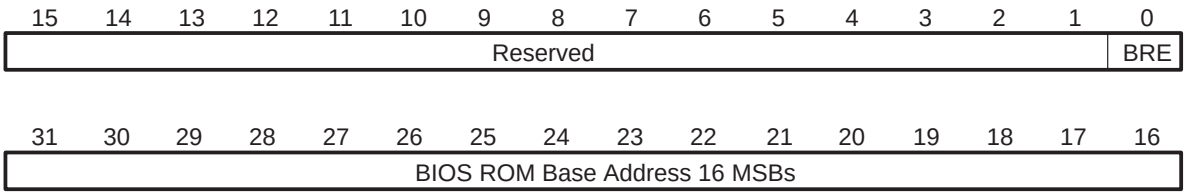
This register holds the base address for ThunderLAN's register set in I/O space. Bit 0 of this register is hardwired to a 1 to indicate that this is a memory-mapped base address. Bits 1 through 3 are hardwired to 0 to indicate that the register set occupies four 32-bit words.

A.1.13 PCI Memory Base Address Register (@ 14h)



This register holds the base address for ThunderLAN's register set in memory space. Bit 0 of this register is hardwired to a 0 to indicate that this is a memory-mapped base address. Bits 1 and 2 are hardwired to 0 to indicate that the register set can be located anywhere in 32-bit address space. Bit 3 of this register (prefetchable bit) is hardwired to a 0, indicating that prefetching is not allowed.

A.1.14 PCI BIOS ROM Base Address Register (@ 30h)



This register holds the base address for ThunderLAN's BIOS ROM in memory space. Bit 0, BRE, is an enable bit for BIOS ROM accesses. When set to a 1, BIOS ROM accesses are enabled; when set to a 0 they are disabled. (The Mem_En in the PCI command register must also be set to allow BIOS ROM accesses.)

A.1.15 PCI NVRAM Register (@ 34h)

This register allows configuration space access to the external EEPROM, in addition to the normal DIO space access through the NetSio register. Control of the EEPROM interface swaps between these two control registers on a most-recently-written basis. Whenever the PCI NVRAM register is written to, it takes control of the EEPROM interface pins. Whenever the DIO_DATA register is written to, the NetSio register takes control of the EEPROM interface

pins. On reset (software or hardware), control of the interface is given to the PCI NVRAM register.

Byte 0

7	6	5	4	3	2	1	0
NVPR	Reserved	DDIR	DATA	Reserved	Reserved	CDIR	CLOCK

Table A–3. PCI NVRAM Register Bits

Bit	Name	Function
7	NVPR	Nonvolatile RAM present: When this bit is set to a 1, it indicates that an external EEPROM is present. When set to a 0, no EEPROM is present.
6	Reserved	This bit is always be read as 0. Writes to this bit are ignored.
5	DDIR	Data direction: When set to a 1, the EDIO pin is driven with the value of the DATA bit. When set to a 0, the value read from the DATA bit reflects the value on the EDIO pin.
4	DATA	This bit is used to read or write the state of the EDIO pin. When DDIR is set to a 1, EDIO is driven with the value in this bit. When DDIR is set to a 0, this bit reflects the value on the EDIO pin.
3	Reserved	This bit is always be read as 0. Writes to this bit are ignored.
2	Reserved	This bit is always be read as 0. Writes to this bit are ignored.
1	CDIR	Clock direction: When set to a 1, the EDCLK pin is driven with the value of the CLOCK bit. When set to a 0, EDCLK pin is not driven† and the value read from the CLOCK bit reflects the incoming value on the EDCLK pin.
0	CLOCK	Clock bit: This bit is used to read or write the state of the EDCLK pin. When CDIR is set to a 1, EDCLK is driven with the value in this bit. When CDIR is set to a 0, this bit reflects the value on the EDCLK pin.

† The EDCLK pin is not driven when the PCI NVRAM register has control of the interface and the CDIR bit is 0 (default after PCI reset).

A.1.16 PCI Interrupt Line Register (@ 3Ch)

This read/writable byte register is used by POST software to communicate interrupt routing information. The contents of this byte have no direct effect on the adapter operation.

A.1.17 PCI Interrupt Pin Register (@ 3Dh)

The interrupt pin register is a read-only byte register that indicates which PCI interrupt pin the adapter uses. As the adapter is a single function device, it is connected to PINTA#. Therefore, the register is hardwired with a value of 01h.

A.1.18 PCI Min_Gnt (@ 3Eh) and Max_Lat (@ 3Fh) Registers

These byte registers are used to specify the adapter's desired settings for latency timers. For both registers, the value specifies a period of time in units of 250 ns (quarter microsecond).

These registers are loaded from an external serial EEPROM on the falling edge of PCI reset, during autoconfiguration. Should autoconfiguration fail (bad checksum), then these registers are loaded with the default values, which are currently 0x00h.

The Min_Gnt register is used for specifying how long a burst period the device needs (assuming a 33-MHz clock). The Max_Lat register is used for specifying how often the device needs to gain access to the PCI bus.

A.1.19 PCI Reset Control Register (@ 40h)

This register is used to disable automatic software resets. The automatic software reset feature is for machines that do not assert the PCI bus PRST# line during soft (Ctrl-Alt-Del) resets, but still performs diagnostics on PCI bus devices. The automatic software reset feature ensures the adapter is reset (and therefore not performing PCI bus master operations) before diagnostics are started.

Soft resets are detected as the deassertion of the BM_En, Mem_En, and IO_En bits in the PCI command register. This differentiates soft resets from bus master disables, which may take place in some systems.

When a soft reset is disabled and the host machine is rebooted, ThunderLAN must be reconfigured without resetting the adapter.

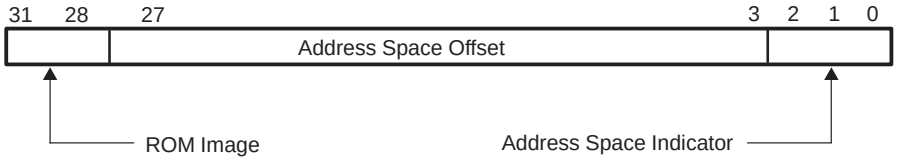
Table A–4. PCI Reset Control Register Bits

Bit	Name	Function
7–1	Reserved	These bits are always read as 0. Writes to these bits are ignored.
0	SRDIS	Soft reset disable: This bit is used to disable automatic software adapter resets. When this bit is set to a 1, an adapter software reset (equivalent to writing a 1 to the Ad_Rst bit) takes place whenever the BM_En, Mem_En, and IO_En bits in the PCI command register are 0. When SRDIS is set to a 1, 0s in these bits do not cause a software reset.

A.1.20 CardBus CIS Pointer (@ 28h)

This register is used by those devices that want to share silicon between CardBus and PCI. The field is used to point to the Card Information Structure (CIS) for the CardBus card. On ThunderLAN this register is hardwired to a value of 10000107h which indicates that the CIS information is in expansion ROM at image1 and offset 20.

For a detailed explanation of the CIS Pointer, refer to the 1995 PC Card Standard Electrical Specification. The subject is covered under the heading CIS Pointer and describes the types of information provided and the organization of this information.

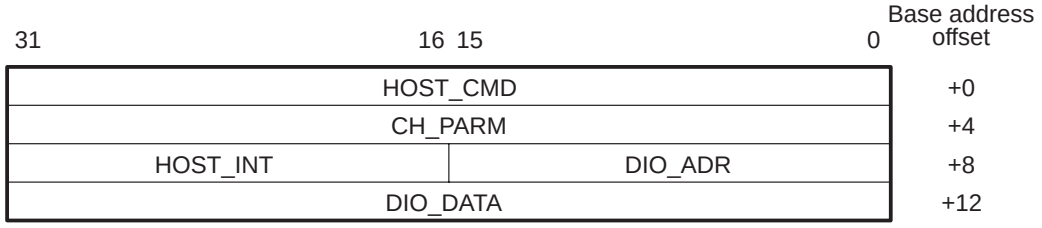


CIS POINTER Layout

A.2 Adapter Host Registers

Host command registers contain bits which are toggled to tell the channel to use receive or transmit FIFOs. ThunderLAN's adapter host registers include the adapter internal registers (see section A.3, Adapter Internal Registers). The following subsections describe the functions of each host register according to protocol.

Figure A–3. Host Interface Address Map



A.2.1 Host Command Register–HOST_CMD @ Base_Address + 0 (Host)

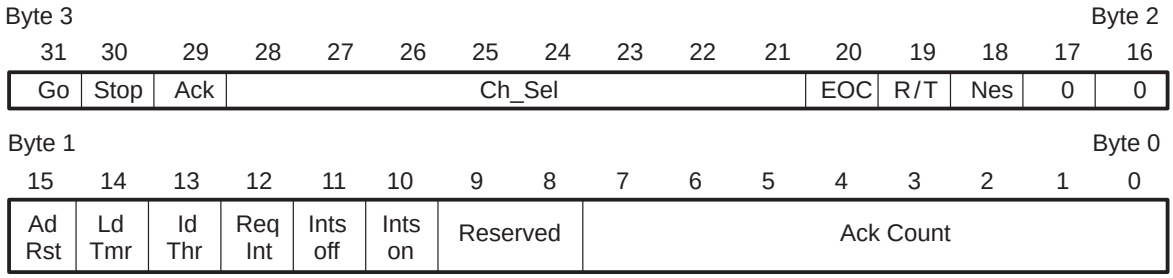


Table A–5. Host_CMD Register Bits

Bit	Name	Function

31	Go	Channel go: This command bit only affects the network channels.
		if R/T = 0 (Tx GO):
		Writing a 1 to this bit starts frame transmission on a stopped or inactive channel. Ch_Parm contains the address of the first transmit list.
		if R/T = 1 (Rx GO):
		Writing a 1 to this bit starts frame reception on a stopped1 or inactive channel. Ch_Parm contains the address of the first receive list.
		Writing a 0 to this bit has no effect. This bit is always read as a 0.

1) Frame transmission and reception are always placed in the stopped (reset) state after reset. therefore, no frames are received into the Rx FIFO and no statistics are logged until the receiver has been started with a GO command.

Table A–5. Host_CMD Register Bits (Continued)

Bit	Name	Function

30	Stop	Channel stop: This command bit only affects the network channels.
		if R/T = 0 (Tx Stop):
		Writing a 1 to this bit stops frame transmission on all transmit channels immediately. All transmit FIFO control logic and the network transmission state machines are placed in a reset state as soon as any ongoing PCI bus transfers are complete (end of current data fragment, list, or CSTAT DMA).
		❑ The TXSTOP[2] bit in the NetSts register is set to indicate that the transmitter has been halted.
		❑ If a STOP is requested during the completion of the DMA of a transmit frame, that frame's interrupts are posted as normal. If that frame was in the last list on the channel, then an EOC interrupt is posted as normal.
		if R/T = 1 (Rx Stop):
		Writing a 1 to this bit stops frame reception on the receive channels immediately. All receive FIFO control logic and the network reception state machines are placed in a reset state as soon as any ongoing PCI bus transfers are complete (end of current data fragment, list, or CSTAT DMA).
		❑ The RXSTOP[3] bit in the NetSts register is set to indicate that the receiver has been halted. While the receiver is stopped, no network Rx statistics are gathered, as the Rx state machines are in reset.
		❑ If a STOP is requested during the completion of the DMA of a transfer frame, that frame's interrupts are posted as normal. If that frame was in the last list on the channel, then an EOC interrupt is posted as normal.
		Writing a 0 to this bit has no effect. This bit is always read as 0.

2) The TXSTOP bit is not set if both transmit channels are already idle. The transmitter will, however, be reset. Because of this, the host must check for EOC or STOP interrupts, in case the transmitter EOCed just as the STOP command was issued. This window is not desirable, and should be fixed on PG2.0

3) The RXSTOP bit is not set if the receive channel is already idle. The receive is, however, reset. Because of this, the host must check for EOC or STOP interrupts, in case the receiver EOCed just as the STOP command was issued. This is not desirable and should be fixed at PG 2.0

Table A–5. Host_CMD Register Bits (Continued)

Bit	Name	Function

29	Ack	Interrupt acknowledge: Writing a 1 to this bit acknowledges the interrupt indicated by the Nes, EOC, Ch_Sel, and R/T fields.
		if Nes = 0, EOC = 1, and R/T = 1 (Status Ack):
		Writing a 1 to this bit acknowledges and clears the status interrupt.
		if Nes = 0, EOC = 0, and R/T = 1 (Statistics Ack):
		Writing a 1 to this bit acknowledges and clears the statistics interrupt.
		if Nes = 1, EOC = 1, and R/T = 0 (Tx EOC Ack):
		Writing a 1 to this bit acknowledges and clears a Tx EOC interrupt for the channel indicated in Ch_Sel.
		if Nes = 1, EOC = 1, and R/T = 1 (Rx EOC Ack):
		Writing a 1 to this bit acknowledges and clears a Rx EOC interrupt for the channel indicated in Ch_Sel.
		if Nes = 1, EOC = 0, and R/T = 0 (Tx EOF Ack):
		Writing a 1 to this bit acknowledges and clears 1 or more Tx EOF interrupts for the channel indicated in Ch_Sel, as indicated in the Ack Count field. If an attempt is made to acknowledge more EOFs than the adapter has outstanding, an AckErr adapter check is raised.
		if Nes = 1, EOC = 0, and R/T = 1 (Rx EOF Ack):
		Writing a 1 to this bit acknowledges and clears 1 or more Rx EOF interrupts for the channel indicated in Ch_Sel, as indicated in the Ack Count field. If an attempt is made to acknowledge more EOFs than the adapter has outstanding, an AckErr adapter check is raised.
		Because of the internal calculations required in Tx EOF and Rx EOF acknowledges, the HOST_INT register is not updated immediately. A short delay (six PCI cycles) is required before reading HOST_INT for such acknowledges to take effect.
		Writing a 0 to this bit has no effect. This bit is always read as 0.

28–21	Ch_Sel	Channel select: This read/write field is used to select between channels on a multichannel adapter. This 8-bit field encodes the channel number (0 through 255). As this adapter supports two channels, only the LSBs are implemented. All the MSBs (28 through 22) are hardwired to 0. This field is written in the same cycle as the command bits and allows a command to be issued in a single write cycle.

Table A–5. Host_CMD Register Bits (Continued)

Bit	Name	Function
20	EOC	End of channel select: This read/write bit is used to select between the EOC, EOF, and command bit operations. If this bit is set to a 1, then end of channel operations are selected. If set to a 0, EOF operations are selected. This bit is also used to select between status and statistics commands when the Nes bit is set to a 0. When Nes is set to 0 and this bit is set to 1, status commands are selected; otherwise statistics commands are selected. This field is written in the same cycle as the command bits, and allows a command to be issued in a single write cycle.
19	R/T	Rx/Tx select: This read/write bit is used to select between receive and transmit command bit operations. If this bit is set to a 1, then receive command operations are selected. if set to a 0, then transmit operations are selected. This field is written to in the same cycle as the command bits, and allows a command to be issued in a single write cycle.
18	Nes	Not error/statistics: This read/write bit is used to select between status (error)/statistics operations and channel operations. If this bit is set to a 0, then status/statistics operations are selected. If set to a 1, channel operations are selected. This field can be written in the same cycle as the command bits, and allows a command to be issued in a single write cycle.
17	0	This bit is always read as 0. Writes to this bit are ignored.
16	0	This bit is always read as 0. Writes to this bit are ignored.
15	Ad_Rst	Adapter reset: Writing a 1 to this bit position causes the adapter to be reset. All channels are cleared and initialized, the ThunderLAN controller is reset, and any network operations are terminated. The receiver and transmitter are placed in their stopped states. The ThunderLAN controller's own (sticky) reset bit is also set. This bit is always read as a 0. There is no need to clear this bit. If this bit is written as a 1 and and the Ints_on and Ints_off bits are also set to 1s, then autoconfiguration of the configuration space registers from the configuration EEPROM takes place.

Table A–5. Host_CMD Register Bits (Continued)

Bit	Name	Function
14	Ld_Tmr	Load interrupt timer[4]: Writing a 1 to this bit causes the interrupt holdoff timer to be loaded from the Ack Count field. Ack Count indicates the time-out period in 4-µs units (based on a 33-MHz PCI clock). The interrupt holdoff timer is used to pace interrupts to the host. Host interrupts are disabled (PCI interrupt request line deasserted) for the time-out period of the timer after an Ack bit write. The timer is restarted on every Ack bit write.
13	Ld_Thr	Load Tx interrupt threshold[5]: Writing a 1 to this bit causes selected transmit channel interrupt threshold to be loaded from the Ack count field. For this operation to be valid (take effect), the following parameters must be set: Nes = 0, EOC = 0, R/T = 0, and Ch_Sel must indicate the selected transmit channel. Set Ack count to indicate the Tx interrupt threshold in number of EOF interrupts. This threshold determines the number of Tx EOF interrupts that must occur before a Tx EOF interrupt is posted. If the threshold is set to 0, no Tx EOF interrupts are posted.
12	Req_Int	Request host interrupt: Writing a 1 to this bit creates a dummy ThunderLAN interrupt. Writing a 0 to this bit has no effect. This bit is always read as 0.
11	Ints_off	Turn PCI interrupts off: Writing a 1 to this bit disables ThunderLAN interrupts to the host (PINTA# will never be asserted). Writing a 0 to this bit has no effect. This bit is always read as 0.
10	Ints_on	Turn PCI interrupts on: Writing a 1 to this bit reenables ThunderLAN interrupts after an Ints_off command bit write. Writing a 0 to this bit has no effect. This bit is always read as 0.
9–8	0	These bits are reserved and are always written as 0s.
7–0	Ack count	EOF acknowledge count: This field is primarily used to acknowledge Rx or Tx EOF interrupts in conjunction with the Ack command bit. It is set with the number of frames (lists) serviced by the host interrupt routine in response to the interrupt. The adapter then uses this parameter to determine if the host has serviced all outstanding frame interrupts, and therefore, whether to generate a further EOF interrupt. This field is also used to pass interrupt threshold and timer values in conjunction with the Ld_Tmr and Ld_Thr command bits.

4) The interrupt timer value may be reloaded at any time. If the timer had already timed out, it remains that way. If it has not yet timed out, it does so when the new timer value is reached or surpassed.

5) The threshold value may be reloaded at any time. Increasing the threshold value causes a current interrupt to be deasserted if the old threshold value has been reached and the new threshold hasn't.

A.2.2 Channel Parameter Register–CH_PARM @ Base_Address + 4 (Host)

This is used to pass parameter information for HOST_CMD register commands as follows:

- GO (Tx GO): Load CH_PARM with the address of the first transmit list before issuing the command. The list must be located on an 8-byte address boundary (three LSBs must be 0).
- GO (Rx GO): Load CH_PARM with the address of the first receive list before issuing the command. The list must be located on an 8-byte address boundary (three LSBs must be 0).

Load CH_PARM and write the GO bit with no intervening DIO accesses, as the DIO_DATA and CH_PARM registers share a common holding register. Driver writers must ensure that the code that loads CH_PARM and writes the GO bit is noninterruptable. Otherwise, an intervening DIO could occur, corrupting the list address and resulting in unpredictable operation.

When read, the LSB of the CH_PARM register indicates the operating mode of the PCI output buffers (5 V or 3 V) for diagnostic purposes. A value of 1 indicates 5-V operation, a value 0 indicates 3-V operation.

In an adapter check, CH_PARM contains the cause of the adapter check (refer to Adapter Check Interrupt. Int_type = 110b and Int_Vec $\neq$ 00h, subsection 4.4.8).

A.2.3 Host Interrupt Register–HOST_INT @ Base_Address + 10 (Host)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	Int Vec								Int Type			0	0

Table A–6. HOST_INT Register Bits

Bit	Name	Function
15–13	0	These bits are always read as 0s.
12–5	Int_Vec	Interrupt vector: This field indicates the highest active interrupt flag for a particular interrupt type. Its format depends on the value read in the Int_type bit. This field is read only.
4–2	Int_Type	Interrupt type: This field indicates the type of interrupt, and therefore, the format of the Int_Vec field. The three bits are coded as follows: 000: No interrupt (invalid code) 001: Tx EOF 010: Statistics overflow 011: Rx EOF 100: Dummy interrupt (created by Req_Int command bit) 101: Tx EOC 110: Adapter check/network status If Int_Vec is 0, this is a network status interrupt. If Int_Vec is not 0, this is an adapter-check interrupt. 111: Rx EOC This field is read only, but writing a nonzero value to it causes the adapter PCI interrupt to be disabled (deasserted) until after a 1 is written to the Ack command bit (in HOST_CMD). If this register is not written to, it maintains the highest priority interrupt that is available, even if the driver is currently servicing a lower priority interrupt.
1	0	This bit is always read as 0.
0	0	This bit is always read as 0.

The HOST_ INT register indicates the reason for a ThunderLAN PCI interrupt. The bit positions in this register are arranged so that its contents may be used as a table offset in a jump table to allow a quick jump to the appropriate interrupt service routine.

The 16 MSB positions in the HOST_CMD register have a one-for-one correspondence with those in the HOST_INT register. The Int_Vec field corresponds to the Ch_Sel field, and the Int_type field corresponds to the EOC, R/T,

and Nes bits. This allows the value read from the interrupt register to be written to the HOST_CMD register to directly select the appropriate channel. If no interrupts are active, the interrupt pacing timer is running, or the PCI interrupt has been disabled (by writing a nonzero value to this register), the HOST_INT register is read as all 0s. If this value (0) is written to HOST_CMD (with the Ack bit set) no interrupts are acknowledged but the interrupt pacing timer is restarted.

A.2.4 DIO Address Register–DIO_ADR @ Base_Address + 8 (Host)

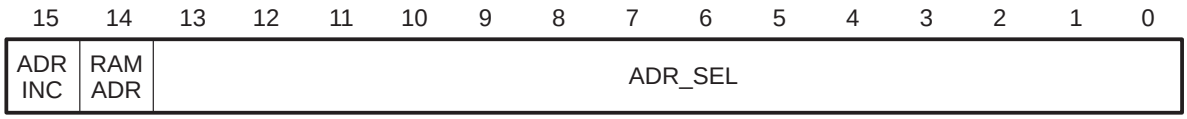


Table A–7. DIO_ADR Register Bits

Bit	Name	Function
15	ADR_INC	Address increment: When this bit is set to a 1, the DIO address in ADR_SEL autoincrements by 4 on each DIO_DATA access. When this bit is set to 0, the DIO address is not affected by accesses to DIO_DATA.
14	RAM_ADR	RAM address select: When this bit is set to 1, DIO accesses are to the muxed SRAM. If this bit is set to 0, DIO accesses are to internal ThunderLAN registers. The internal muxed SRAM can only be accessed while the adapter is in reset (NRESET bit in the NetCmd register is set to a 0). Accesses to the SRAM while the adapter is not in reset are undefined; they are ignored and return unknown data.
13–0	ADR_SEL†	This field contains the DIO address, the SRAM or the internal register address to be used on subsequent accesses to the DIO_DATA register. If the ADR_INC bit is set, this field autoincrements by 4 on accesses to the DIO_DATA register. ❑ For register accesses, the seven MSBs [13::7] of ADR_SEL are ignored. The seven LSBs [6::0] indicate the byte address of the register, but the two LSBs are not used since byte control is through the PCI bus byte enables. ❑ For RAM, accesses to the three MSBs [13::11] of ADR_SEL are ignored. The 11 LSBs [10::0] indicate the RAM row address [10::2] and RAM word address [1::0].

† This is a byte address.

RAM Addressing

The RAM is composed of 431, 68-bit words. Bits [10::2] of ADR_SEL indicate the ROW address. Bits [1::0] are used to indicate which part of the 68-bit word is to be accessed.

- If ADR_SEL[1::0] = 00, the 32 LSBs of the 68-bit word are accessed.
- If ADR_SEL[1::0] = 01, the middle 32 bits of the 68-bit word are accessed.
- If ADR_SEL[1::0] = 1X, the four MSBs of the 68-bit word are accessed (in the four LSBs of DIO_DATA).

PCI bus-byte enables are honored in writes to the internal RAM; individual byte writes are allowed. If autoincrement mode is enabled, only the row address increments; the word address is not affected.

A.2.5 DIO Data Register – DIO_DATA @ Base_Address + 12 (Host)

The DIO_DATA register address allows indirect access to internal ThunderLAN registers and SRAM. There is no actual DIO_DATA register; accesses to this address are mapped to an internal bus access at the address specified in the DIO_ADR register.

The DIO_DATA location uses 32-bit PCI data transfers with full-byte control, following the normal *PCI Local Bus Specification* conventions. ThunderLAN uses the target-ready (PTRDY#) signal to insert PCI wait states and to ensure correct data transfers.

Writes to this register cause control of the EEPROM interface pins to go to the NetSio register. Control of the EEPROM interface swaps between the PCI NVRAM register and the NetSio register on a most-recently-written basis. Whenever the PCI NVRAM register is written to, it takes control of the EEPROM interface pins. Whenever the DIO_DATA register is written to, the NetSio register takes control of the EEPROM interface pins.

A.3 Adapter Internal Registers

The adapter's internal registers are indirectly accessible from the PCI bus through the DIO_ADR and DIO_DATA registers. These are usually referred to as DIO. ThunderLAN has an internal 32-bit bus that is used for DIO accesses to the registers and the SRAM. PCI bus byte enables are maintained on this internal bus, allowing arbitrary byte transfers.

DIO accesses are primarily used to initialize and control the ThunderLAN controller. Normally, only ThunderLAN controller registers are accessible by DIO. In adapter test mode (PCI configuration option), the SRAM, FIFO control registers, and some PCI controller registers are also accessible.

The content and meaning of some registers vary with the network protocol in use. The following subsections describe the functions of each register according to the protocol.

Figure A–4. ADAPTER Internal Register Map

Byte 3	Byte 2	Byte 1	Byte 0	DIO Address
NetMask	NetSts	NetSio	NetCmd	0x00
ManTest		NetConfig		0x04
Default device ID MSbyte	Default device ID LSbyte	Default vendor ID MSbyte	Default vendor ID LSbyte	0x08
Default Max_Lat	Default Min_Gnt	Default subclass	Default revision reg	0x0C
Areg_0 (23 to 16)	Areg_0 (31 to 24)	Areg_0 (39 to 32)	Areg_0 (47 to 40)	0x10
Areg_1 (39 to 32)	Areg_1 (47 to 40)	Areg_0 (7 to 0)	Areg_0 (15 to 8)	0x14
Areg_1 (7 to 0)	Areg_1 (15 to 8)	Areg_1 (23 to 16)	Areg_1 (31 to 24)	0x18
Areg_2 (23 to 16)	Areg_2 (31 to 24)	Areg_2 (39 to 32)	Areg_2 (47 to 40)	0x1C
Areg_3 (39 to 32)	Areg_3 (47 to 40)	Areg_2 (7 to 0)	Areg_2 (15 to 8)	0x20
Areg_3 (7 to 0)	Areg_3 (15 to 8)	Areg_3 (23 to 16)	Areg_3 (31 to 24)	0x24
HASH2				0x28
HASH1				0x2C
Tx underrun	Good Tx frames			0x30
Rx overrun	Good Rx frames			0x34
Code error frames	CRC error frames	Deferred Tx frames		0x38
Single collision Tx frame		Multicollision Tx frames		0x3C
Acommit	Carrier loss errors	Late collisions	Excessive collisions	0x40
MaxRx		BSIZEreg	LEDreg	0x44

A.3.1 Network Command Register–NetCmd @ 0x00 (DIO)

All bits in this register are set to 0 on an Ad_Rst or when PRST# is asserted.

Byte 0

7	6	5	4	3	2	1	0
NRESET	NWRAP	CSF	CAF	NOBRX	DUPLEX	TRFRAM	TXPACE

Table A–8. Network Command Register Bits

Bit	Name	Function
7	NRESET	ThunderLAN controller not reset: This bit is set to a 1 or a 0 by DIO. This bit is set to a 0 (active) by an Ad_Rst or a PCI reset. When this bit is set to a 0, the ThunderLAN controller is kept in a reset state. Network configuration in the NetConfig register can only be altered while this bit is set to a 0.
6	NWRAP	Not wrap: This bit determines the flow of data to and from the adapter. This bit is set to a 0 (active) by an Ad_Rst or by a PCI reset. If this bit is set to a 0, data is internally wrapped by the adapter and does not pass through the MII or internal PHY. If this bit is set to 1, network Rx and Tx data pass the selected network PHY. In internal wrap mode, the adapter media access control (MAC) logic is clocked by the PCI bus clock. All network clocks are ignored, and all MII pins, except MDCLK/MDIO, are placed in the high-impedance state.†
5	CSF	Copy short frames/copy routed frames: The function of this bit depends on the MAC protocol in use: ❑ CSMA/CD mode: If this bit is set, the receiver does not discard frames that are shorter than a slottime (64 bytes). ❑ Demand priority/token ring mode: This bit has no function.
4	CAF	Copy all frames: When this bit is set to 1, the ThunderLAN controller receives frames indiscriminantly, without regard to destination address. CAF does not copy error frames unless the PEF bit in the NetConfig register is set.
3	NOBRX	No broadcast Rx: When this bit is set to 1, the ThunderLAN controller does not receive frames with broadcast addresses (such as 0xffff.ffff.ffff, the all nodes broadcast address; or 0xC000.ffff.ffff, the token ring all nodes for this ring broadcast address). When this bit is set to 0, broadcast frames are received by the adapter.
2	DUPLEX	Full duplex: When this bit is set to 1‡, the ThunderLAN controller transmits and receives frames simultaneously in full duplex. When this bit is set to 0, the ThunderLAN controller transmits and receives frames in half duplex.
1	TRFRAM	Token ring frame format: When this bit is set to 1, the ThunderLAN controller uses the token ring frame format for all frames transmitted or received. When this bit is set to 0, the ThunderLAN controller uses the Ethernet frame format for all frames transmitted or received.

† Because this bit directly switches the network logic clocks, it should only be changed while NRESET is active.
‡ Collision statistics are disabled in full duplex.

Table A–8. Network Command Register Bits (Continued)

Bit	Name	Function
0	TXPACE	Transmit pacing (CSMA/CD): This bit allows pacing of transmitted CSMA/CD frames to improve network utilization of network file servers. When this bit is set, the pacing algorithm is enabled. When this bit is cleared, the pacing algorithm is disabled. The pacing algorithm automatically delays new adapter frame transmissions in contention situations. If a transmitted frame either collides with another frame or has to defer to another transmission, a pacing delay of four interframe gaps (4*96 bit-times) is inserted between new frame transmissions. This pacing delay continues to be inserted until 31 sequential frames are transmitted without collision or deference.

A.3.2 Network Serial I/O Register – NetSio @ 0x00 (DIO)

This register shares control of the external EEPROM interface with the PCI NVRAM register. Control of the EEPROM interface swaps between these two control registers on a most-recently-written basis. Whenever the PCI NVRAM register is written to, it takes control of the EEPROM interface pins. Whenever the DIO_DATA register is written to, the NetSio register takes control of the EEPROM interface pins. On reset (software or hardware), control of the interface is given to the PCI NVRAM register. All bits in this register are set to 0 on an Ad_Rst or when PRST# is asserted.

Byte 1

15	14	13	12	11	10	9	8
MINTEN	ECLOK	ETXEN	EDATA	NMRST	MCLK	MTXEN	MDATA

Table A–9. Network Serial I/O Register Bits

Bit	Name	Function
15	MINTEN	MII Interrupt enable: When this bit is set to 1, the MIRQ interrupt bit is set if the MDIO pin is asserted low.
14	ECLOK	EEPROM SIO clock: This bit controls the state of the EDCLK pin. When this bit is set to 1, EDCLK is asserted. When this bit is set to 0, EDCLK is deasserted. This bit is also used to determine the state of the EEPROM interface. If the EEPROM port is disabled, this bit is always read as 0, even if a value of 1 is written to the bit. ThunderLAN detects that the EEPROM port is disabled by sensing the state of the EDCLK pin during reset. If the EDCLK pin is read as 0 during reset (due to an external pulldown resistor), then the EEPROM interface is disabled and no attempt is made to read configuration information.
13	ETXEN	EEPROM SIO transmit enable: This bit controls the direction of the EDIO pin. When this bit is set to 1, EDIO is driven with the value in the EDATA bit. When this bit is set to 0, the EDATA bit is loaded with the value on the EDIO pin.

Table A–9. Network Serial I/O Register Bits (Continued)

Bit	Name	Function
12	EDATA	EEPROM SIO data: This bit is used to read or write the state of the EDIO pin. When ETXEN is set to 1, EDIO is driven with the value in this bit. When ETXEN is set to 0, this bit is loaded with the value on the EDIO pin.
11	NMRST	MII not reset: This bit can be set to 1 or 0 by the DIO. This bit is set to 0 (active) by an Ad_Rst or a PCI reset. The state of this bit directly controls the state of the MRST# (MII reset) pin. If this bit is set to 0, the MRST# pin is asserted. If this bit is set to 1, the MRST# pin is deasserted. This bit is not self-clearing and must be manually deasserted. It can be set low and then immediately set high. Note that since every PHY attached to the MII may not have a reset pin, you need to both do NMRST and individually reset each PHY.
10	MCLK	MII SIO clock: This bit controls the state of the MDCLK pin. When this bit is set to 1, MDCLK is asserted. When this bit is set to 0, MDCLK is deasserted.
9	MTXEN	MII SIO transmit enable: This bit controls the direction of the MDIO pin. When this bit is set to 1, MDIO is driven with the value in the MDATA bit. When this bit is set to 0, the MDATA bit is loaded with the value on the MDIO pin.
8	MDATA	MII SIO data: This bit is used to read or write the state of the MDIO pin. When MTXEN is set to 1, MDIO is driven with the value in this bit. When MTXEN is set to 0, this bit is loaded with the value on the MDIO pin.

A.3.3 Network Status Register – NetSts @ 0x00 (DIO)

All bits in this register are set to 0 on an Ad_Rst or when PRST# is asserted.

Byte 2

23	22	21	20	19	18	17	16
MIRQ	HBEAT	TXSTOP	RXSTOP	Reserved			

Table A–10. Network Status Register Bits

Bit	Name	Function
23	MIRQ	MII interrupt request: This bit is set whenever ThunderLAN detects that the MDIO pin is asserted low and the MINTEN bit in the NetSio register is set. Assertion low of the MDIO line between MII control frames is an indication from the PMI/PHY of an error or a line status change. This bit is cleared by writing a 1 to its bit position. Writing a 0 has no effect.
22	HBEAT	Heartbeat error: In CSMA/CD mode, a heartbeat interrupt is posted if MCOL is not asserted during the interframe gap following frame transmission. This bit is cleared by writing a 1 to its bit position. Writing a 0 has no effect.
21	TXSTOP	Transmitter stopped: This bit indicates the completion of a transmit STOP command. This bit is cleared by writing a 1 to its bit position. Writing a 0 has no effect.

Table A–10. Network Status Register Bits (Continued)

Bit	Name	Function
20	RXSTOP	Receiver stopped: This bit indicates the completion of a receive STOP command. This bit is cleared by writing a 1 to its bit position. Writing a 0 has no effect.
19–16	Reserved	

A.3.4 Network Status Mask Register–NetMask @ 0x00 (DIO)

This register determines whether network status flags in the NetSts register cause interrupts or not. Each bit in this register acts as a mask on the corresponding NetSts register bit. If the mask bit is set, then an interrupt is raised if the corresponding status flag is set. All bits in this register are set to 0 on an Ad_Rst or when PRST# is asserted.

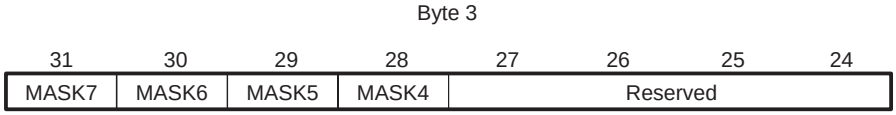


Table A–11. Network Status Mask Register Bits

Bit	Name	Function
31	MASK7	MII interrupt mask: When this bit is set, a network status interrupt is posted if the MIRQ bit in the NetSts register is set.
30	MASK6	Heartbeat error mask: When this bit is set, a network status interrupt is posted if the HBEAT bit in the NetSts register is set.
29	MASK5	Transmit stop mask: When this bit is set, a network status interrupt is posted if the TXSTOP bit in the NetSts register is set.
28	MASK4	Receive stop mask: When this bit is set, a network status interrupt is posted if the RXSTOP bit in the NetSts register is set.
27–24	Reserved	

A.3.5 Network Configuration Register – NetConfig @ 0x04 (DIO)

This 16-bit register is used for ThunderLAN's controller configuration. This register is only writable while the ThunderLAN controller is in reset. (NRESET = 0). All bits in this register are set to 0 on an Ad_Rst or when PRST# is asserted.

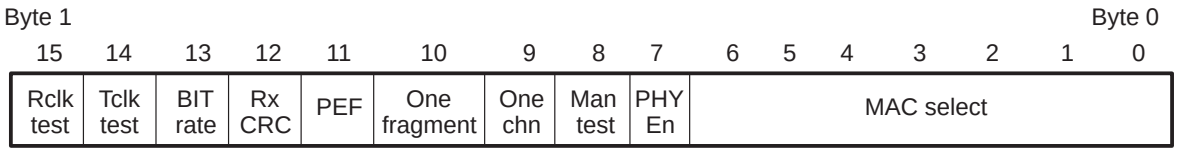


Table A–12. Network Configuration Register Bits

Bit	Name	Function
15	Rclk test	Test MRCLK: This test/sense bit allows the host to verify the presence of a clock on the MRCLK pin. This bit can only be written as a 1. Writing a 0 to this bit has no effect. This bit is cleared to 0 by a rising edge on MRCLK. Host software verifies a minimum clock frequency by setting this bit, waiting for the maximum clock period, and then verifying the bit has been cleared.
14	Tclk test	Test MTCLK: This test/sense bit allows the host to verify presence of a clock on the MTCLK pin. This bit can only be written as a 1. Writing a 0 to this bit has no effect. This bit is cleared to 0 by a rising edge on MTCLK. Host software verifies a minimum clock frequency by setting this bit, waiting for the maximum clock period, and then verifying the bit has been cleared.
13	BITrate	Bit rate MII: When this bit is set to 1, ThunderLAN supports a bit-level (rather than nibble-level) CSMA/CD MII. This mode is only valid for 10M-bps operation. The MII pins can be tied to a standard Ethernet SNI that follows the NS8391 interface. In this mode, the MII pins should be connected as follows: ❑ MRXD0 to RXD (receive data) ❑ MRCLK to RXC (receive clock) ❑ MCRS to CRS (carrier sense) ❑ MTCLK to TXC (transmit clock) ❑ MTXD0 to TXD (transmit data) ❑ MTXEN to TXE (transmit enable) ❑ MCOL to COL (collision detect) ❑ MTXD3, MTXD2, MTXD1, and MTXER are driven with values contained in the low nibble (bits 0–3) of the Acommit register. These signals can be used to drive PHY option pins, such as loopback or UTP/AUI select.

Table A–12. Network Configuration Register Bits (Continued)

Bit	Name	Function
12	RxCRC	Receive CRC: When this bit is set to 1, the ThunderLAN controller transfers frame CRC to the host for received frames and includes it in the reported frame length. The value in the MaxRx register must be large enough to accommodate the data field plus this transferred CRC. Failure to do so may result in an Rx overrun
11	PEF	Pass error frames: When this bit is set to 1 and the CAF bit is set in NetCmd (copy all frames mode), frames that would otherwise be rejected by the adapter due to CRC, alignment, or coding errors are passed to the host. Such frames have the Rx_Error bit in the list receive CSTAT set to differentiate them from good frames. This mode does not affect adapter frame statistics.
10	One fragment	One fragment mode: When this bit is set to 1, the adapter only reads a single fragment (data count/data address pair) on the receive channel (rather than ten). When this bit is set to 0, the adapter reads and uses up to ten fragments. If multifragment mode is selected and less than ten fragments are used, the host must write a 0 to the data count field of the fragment immediately after the last fragment used. Generally, receive frames are not fragmented because they must be examined by host software first. One fragment mode can be used in such cases, where it improves adapter performance by reducing the number of PCI cycles needed to read the receive list.
9	One chn	One channel mode: When this bit is set to 1, the adapter only supports a single transmit channel (rather than two). When this bit is set to 0, the adapter supports two transmit channels. The adapter has 3K bytes of FIFO RAM for frame buffering. In normal two-channel mode, 1.5K is allocated to Rx (one Rx channel), and 1.5K to transmit (0.75K per Tx channel). In one-channel mode, all 1.5K of Tx FIFO RAM is allocated to channel 0, the only channel. In one-channel mode, the HOST_CMD bit operations on channel 1 are ignored.
8	MTEST	Manufacturing test: When this bit is set to 1, the adapter is placed into manufacturing test mode. Manufacturing test mode is reserved for Texas Instruments manufacturing test. Operation of the adapter with this bit set is undefined.
7	PHY_En	On-chip PHY enable: This bit is used to enable/disable the adapter's on-chip 10Base-T PHY. When this bit is set to 0, the on-chip PHY is disabled and placed in a powered-down state. When this bit is set to a 1, the on-chip PHY is enabled as a potential PHY on the shared MII (it still needs to be selected/configured using MDCLK/MDIO).†
6–0	MAC select	MAC protocol select: This field is used to select the network MAC protocol required. The seven-bit code allows for 128 unique network configurations. This version of ThunderLAN supports four protocol modes: CSMA/CD and three types of data stream interfaces (supporting external demand priority and token ring implementations). Bits 6 through 2 are, therefore, hardwired to 0.

† If the on-chip PHY is selected as the active MII PHY, all MII pins except MDCLK and MDIO are disabled.

Table A–13. MAC Protocol Selection Codes

Code	MAC Protocol Selected
0xb0000000b	CSMA/CD (802.3 -10/100M bps)
0b0000001b	External protocol: Enhanced 802.3u interface for 802.12 – 100M bps ❑ 100VG-AnyLAN interface, with decreased priority determined by channel ❑ Tx start-up timing hardwired at 50 cycles
0xb0000010b	External protocol: Enhanced 802.3u interface for 802.12 – 100M bps ❑ 100VG-AnyLAN interface, with DP priority determined by frame CSTAT ❑ Tx start-up timing hardwired at 50 cycles
0xb0000011b	External protocol: Enhanced 802.3u interface for any network ❑ Priority determined by frame CSTAT ❑ Tx timing controlled by external device
0xb0000100b – 0xb1111111b	Reserved

A.3.6 Manufacturing Test Register – ManTest @ 0x04 (DIO)

This 16-bit register is used for manufacturing test. The options controlled by this register only take effect while the MTEST bit in the NetConfig register is set. The functions controlled by this register are for Texas Instruments manufacturing test only.

A.3.7 Default PCI Parameter Registers – @ 0x08 – 0x0C (DIO)

These eight read-only bytes indicate the default contents of the autoloadable PCI configuration registers. These default values are loaded into the respective PCI configuration registers if autoload fails (bad checksum).

Figure A–5. Default PCI Parameter Register

DIO Address	Byte 3	Byte 2	Byte 1	Byte 0
0x08	Default device ID (0x50) MSbyte	Default device ID (0x00) LSbyte	Default vendor ID (0X10)MSbyte	Default vendor ID (0x4c) Msbyte
0x0C	Default Max_Lat (0x00)	Default Min_Gnt (0x00)	Default subclass (0x08)	Default revision† (0x30)

† The value of the Default revision register has changed from 23h for PG2.3 to 30h for PG3.03.

A.3.8 General Address Registers – Areg_0-3 @ 0x10 – 0x24 (DIO)

The four general-purpose address registers, Areg_0 through Areg_3, are used to hold the adapter's specific and group addresses. Each of the four registers can be used to hold any 48-bit IEEE 802 address (specific or group, local or universal).

Each register holds a 48-bit address, and all four registers are directly compared against the destination address field of incoming frames. If any of the four address registers match the incoming address, the frame is copied.

Addresses should be written to the registers in the native data format of the protocol in use; that is, in the same format in which the address field of the received frame appears in memory. As an example, the group/specific bit of an Ethernet-style frame appears in the Areg bit 40 (LSB of MSbyte). In contrast, the same bit in a token ring style frame appears in bit 47 (MSB bit of MSbyte).This is because the Ethernet frame format uses LSB first transmission while the token ring frame format uses MSB first (the specific/group bit is always the first address bit on the wire).

The general address registers have an addressing lockout function that prevents use of any register for address comparison while being written or when it is uninitialized. Addressing lockout is enabled at NRESET or whenever the first (Areg[47::40]) register byte is written and it is disabled whenever the last (Areg[7::0]) register byte is written.

A.3.8.1 The All-Nodes Broadcast Address

In addition to the general address matching of specific or group addresses, the adapter also responds to the all nodes broadcast address of 0xFFFF FFFF FFFF if the NOBRX (no broadcast Rx) bit in NetCmd is not set.

A.3.8.2 Token Ring Frame Format Addressing Extensions

There are a number of extensions to the basic 802 addressing that is used in token ring networks. Token ring provides functional addresses as a subset of the local-group addresses and an all nodes (this ring) broadcast address.

Functional addresses provide bit position-based addressing of common network functions. Bit positions 30 through 0 of the incoming destination address are compared with 31 functional address bits (stored in a register). If any incoming functional bit has its corresponding register bit set, the frame is copied due to a functional address match. Functional addresses are identified from normal local group addresses by having the MSB of the third address byte (the group/functional bit) set to a 0.

The ThunderLAN adapter supports functional addressing, when token ring frame format is selected over the demand priority access method. In this

mode, functional addressing is supported through the general address registers. If any address register contains a functional address (group/specific = 1; local/universal = 1; group/functional = 0), that register's two MSbytes are compared normally, but its 31 LSBs are compared on a functional bit-match basis. Any of the registers can be used to hold functional addresses; all the registers are identical.

In token ring frame format mode, the adapter responds to an additional broadcast address: The all-nodes (this ring) broadcast address of 0xC000 FFFF FFFF (but only if the NOBRX bit in the NetCmd register is not set).

A.3.9 Hash Address Registers – HASH1/HASH2 @ 0x28 – 0x2C (DIO)

The hash registers allow group-addressed frames to be accepted on the basis of a hash function of the address. The hash function calculates a six bit data value from the 48-bit destination address, as follows:

```
Hash_fun(0) = DA(0) xor DA(6)   xor DA(12) xor DA(18) xor
DA(24) xor DA(30) xor DA(36) xor DA(42);

Hash_fun(1) = DA(1) xor DA(7)   xor DA(13) xor DA(19) xor
DA(25) xor DA(31) xor DA(37) xor DA(43);

Hash_fun(2) = DA(2) xor DA(8)m   xor DA(14) xor DA(20) xor
DA(26) xor DA(32) xor DA(38) xor DA(44);

Hash_fun(3) = DA(3) xor DA(9)   xor DA(15) xor DA(21) xor
DA(27) xor DA(33) xor DA(39) xor DA(45);

Hash_fun(4) = DA(4) xor DA(10)   xor DA(16) xor DA(22)
xorm DA(28) xor DA(34) xor DA(40) xor DA(46);

Hash_fun(5) = DA(5) xor DA(11) xor DA(17) xor DA(23) xor
DA(29) xor DA(35) xor DA(41) xor DA(47);
```

These bits are used as an offset to a 64-bit table, which indicates whether to copy a given frame or not. This table is stored in the two HASH registers, a bit value of 1 indicating a frame should be matched, a bit value of 0 that it should not.

HASH1 contains the 32 LSBs of the hash table with entries 0 through 31 mapped to the corresponding register bits, 0 through 31. HASH2 contains the 32 MSBs of the hash table with hash table entries 32 through 63 mapped to register bits 0 through 31.

A.3.10 Network Statistics Registers – @ 0x30 – 0x40 (DIO)

The network statistics registers gather frame error information. Registers vary in size, depending on the frequency with which they increment, and may be 8, 16, or 24 bits wide. Reading a statistics register clears its contents after the read. Byte reads to a multibyte register clear the contents of the bytes read only. As long as such registers are read in natural order (LSbyte first), no statistics will be lost, even when registers are read a byte at a time. Writing to a statistics register has no effect.

The MSBs of all the error counters are ORed together to create the statistics overflow interrupt vector (Int_type = 010) in the HOST_INT register. As more than one counter may have overflowed, all statistics registers must be read (cleared) on a statistics overflow interrupt.

Figure A–6. Ethernet Error Counters

DIO Address	Byte 3	Byte 2	Byte 1	Byte 0
0x30	Tx underrun	Good Tx frames		
0x34	Rx overrun	Good Rx frames		
0x38	Code error frames	CRC error frames	Deferred Tx frames	
0x3C	Single collision Tx frames		Multicollision Tx frames	
0x40		Carrier loss errors	Late collisions	Excessive collisions

Table A–14. Ethernet Error Counters

Counter	Definition
Good Tx frames	are without errors. This is a 24-bit counter. Good frames are transmitted more frequently than errored frames.
Tx frames	are aborted during transmission, due to frame data not being available (due to host bus latencies). This is a byte-wide counter.
Good Rx frame underruns	are received without errors. This is a 24-bit counter. Good frames are received more frequently than errored frames.
Rx overrun frames	are address-matched and could not be received due to inadequate resources (Rx FIFO full) or because their frame size exceeded MaxRx. This is a byte-wide counter.
Deferred Tx frames	were deferred to prior network traffic on their initial attempt at transmission. This is a 16-bit counter.
CRC error frames	are received with CRC errors, but without alignment or coding errors. This is a byte-wide counter.
Code error frames	are received with alignment (not an even number of nibbles) or code errors (MRXER signaled from PHY). This is a byte-wide counter.
Multicollision Tx frames	have encountered 2 to 15 collisions before being transmitted on the network. This is a 16-bit counter.
Single collision frames	have encountered one collision before being transmitted on the network. This is a 16-bit counter.
Excessive collisions	have failed to gain access to the network in 16 attempts (frames that experienced 16 collisions). This is a byte-wide counter.
Late collisions	have been interrupted by a collision after the network slottime. This is a byte-wide counter.
Carrier loss errors	are sent by the adapter and for which a receive carrier was not detected after a slottime from the start of transmission. The carrier must be present continuously from this point until the end of transmission to prevent an error being logged.† This is a byte-wide counter.

† Carrier loss errors are not logged in full-duplex mode.

Figure A–7. Demand Priority Error Counters

DIO Address	Byte 3	Byte 2	Byte 1	Byte 0
0x30	Rx overrun	Good Rx frames		
0x34	Tx underrun	Good Tx frames		
0x38	Code error frames	CRC error frames	Deferred Tx frames	
0x3C				
0x40				

Table A–15. Demand Priority Error Counters

Counter	Definition
Good Tx frames	are transmitted without errors. This is a 24-bit counter. Good frames are transmitted more frequently than errored frames.
Tx underrun frames	are aborted during transmission, due to frame data not being available (due to host bus latencies). This is a byte-wide counter.
Good Rx frames	are received without errors. This is a 24-bit counter. Good frames are received more frequently than errored frames.
Rx overrun frames	are address-matched and could not be received due to inadequate resources (Rx FIFO full). This is a byte-wide counter.
CRC error frames	are received with CRC errors, but without alignment or coding errors. This is a byte-wide counter.
Code error frames	are received with alignment (not an even number of nibbles) or code errors (MRXER signaled from PMI). This is a byte-wide counter.

A.3.11 Adapter Commit Register – Acommit @ 0x40 (DIO) (Byte 3)

The adapter commit register indicates the PCI commit size of the adapter.

Byte 3	31	30	29	28	27	26	25	24
	Tx commit level				PHY options			

Table A–16. Adapter Commit Register Bits

Bit	Name	Function
31–28	Tx commit level	Transmit commit level: This nibble code indicates the commit size in use by the adapter transmitter. The code indicates the number of bytes that must be in a channel's FIFO before network transmission is started. At reset, the commit level is set to 0, giving minimum latency. It is incremented every time a frame is aborted due to a FIFO underrun. The adapter, therefore, automatically adapts itself to the latency available on the host bus. Every increment in level corresponds to a doubling of latency size. The commit levels are: ❑ 0: 64 bytes ❑ 1: 128 bytes ❑ 2: 256 bytes ❑ 3: 512 bytes ❑ 4: 1024 bytes ❑ 5–7: whole frame When the transmit commit level is 3 or greater (512 bytes or more), transmission begins if a FIFO deadlock condition occurs. If the transmitter is waiting for required data in the FIFO and the next PCI data transfer is waiting for room to be freed up in the FIFO before it starts, a deadlock situation exists and transmission never starts. The deadlock is broken by detecting this condition and allowing network transmission to proceed before the full commit level is reached. This situation only occurs where large commit levels are combined with large fragment, burst, and frame sizes.
27–24	PHY options	When ThunderLAN is configured for a bit-rate CSMA/CD MII (BITrate option bit in the NetConfig register), the contents of these bits are presented on the MTXD[3::1] and MTXER pins to allow selection of PHY options. Pin mapping is as follows: ❑ Bit 27 – MTXD3 (full duplex disable) ❑ Bit 26 – MTXD2 (loopback enable) ❑ Bit 25 – MTXD1 (10Base-T (0)/AUI-ThinNet (1) select) ❑ Bit 24 – MTXER (reserved (0))

All bits in this register are set to 0 on an Ad_Rst or when PRST# is asserted. The MSnibble of this register can be written to only when the adapter is in reset (NRESET bit is set to 0).

A.3.12 LED Register–LEDreg @ 0x44 (DIO) (Byte 0)

This byte register contains the value that is driven on the EAD pins whenever BIOS ROM accesses are not taking place (when EXLE, EALE and EOE# are all inactive). Light emitting diodes (LEDs) connected to the EAD pins (directly or buffered) can be controlled by software through this register. All bits in this register are set to 0 on an Ad_Rst or when PRST# is asserted. The values that are output on the EAD pins are the inverse of those which are written to LEDreg.

A.3.13 Burst Size Register–BSIZEreg @ 0x44 (DIO) (Byte 1)

This register is used to set the receive and transmit burst sizes to be used by the adapter. This register is only writable while the ThunderLAN controller is in reset. (NRESET = 0). This register is set to 0x22 on an Ad_Rst or when PRST# is asserted.

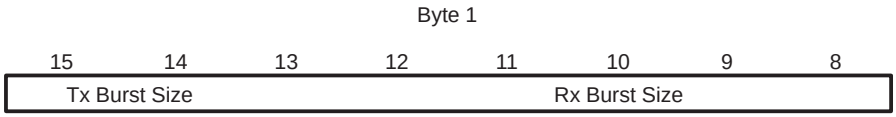
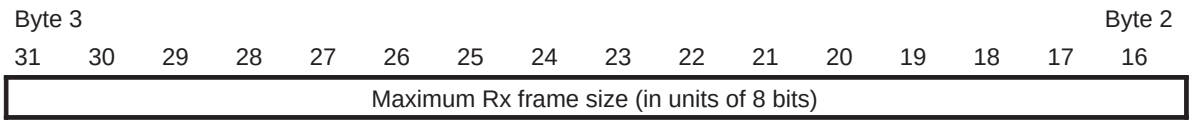


Table A–17. Burst Size Register Bits

Bit	Name	Function
15–12	Tx burst size	Transmit burst size: This nibble code indicates the burst size to be used in data transfers for transmit operations (Tx list or data transfers). The code indicates the maximum number of bytes to be transferred in any one transmit DMA data burst. At reset, the burst size is set to a default level of 64 bytes (code = 2). The burst size codes are: ❑ 0: 16 bytes ❑ 1: 32 bytes ❑ 2: 64 bytes (default) ❑ 3: 128 bytes ❑ 4: 256 bytes ❑ 5–7: 512 bytes ❑ 8–F: reserved
11–8	Rx burst size	Receive burst size: This nibble code indicates the burst size to be used in data transfers for receive operations (Rx list or data transfers). The code indicates the maximum number of bytes to be transferred in any one receive DMA data burst. At reset, the burst size is set to a default level of 64 bytes (code = 2). The burst size codes are: ❑ 0: 16 bytes ❑ 1: 32 bytes ❑ 2: 64 bytes (default) ❑ 3: 128 bytes ❑ 4: 256 bytes ❑ 5–7: 512 bytes ❑ 8–F: reserved

A.3.14 Maximum Rx Frame Size Register – MaxRx @ 0x44 (DIO) (Bytes 2+3)



This register is used to set the maximum size of received network frames. Frames larger than this size are not copied and are counted as Rx-overrun error frames. Setting this parameter prevents an oversize frame from overrunning the buffer space allocated in its receive list and, thereby, causing an adapter check.

If you chose to pass the four-byte CRC field along with your data by asserting the RxCRC in the NetConfig register, MaxRx should be large enough to accommodate this additional field. In this case, the CRC is treated exactly as the data, and failure to accommodate this field could result in frames not being copied and an Rx overrun.

The register holds the maximum frame size in bytes. The value in the register indicates the maximum frame size that can be accommodated in the host buffers. Frames larger than this value are discarded, except in PEF mode. A value of 0 in this register (default) is equivalent to a maximum frame size of 64K bytes; the adapter attempts to receive all frames.

In pass-errored-frame mode, oversize frames are written to the host. The frame size reported in the list CSTAT field is the size of the buffer, not the size of the frame received. The last eight bytes written are corrupt and should be ignored.

For example:

If a 133-byte frame is received in PEF mode with MaxRx set to 128, then:

- ❏ The CSTAT frame length indicates 128 bytes
- ❏ The first 120 bytes in the buffer contain the first 120 bytes of the frame
- ❏ The contents of the last eight bytes is not important

It should be noted that frames that exceed MaxRx are always counted as Rx overrun errors, regardless of PEF mode.

A.3.15 Interrupt Disable Register - INTDIS @ 0x48 (DIO) (BYTE 0)

This register is used to disable RX EOC, RX EOF and TX EOC interrupts. TX EOF can be disabled by setting to Tx interrupt threshold value to a zero. This register is only written to while the ThunderLAN Controller is reset. (NRESET=0)

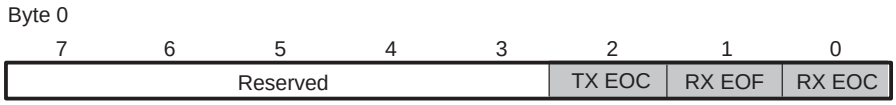


Table A–18. Demand Priority Error Counters

Bit	Name	Function
7 – 3		Reserved
2	TX EOC†	Disable Transmit End of Channel Select: When this bit is set to 1, all transmit channels of TX EOC interrupts are disabled. Default value is 0.
1	RX EOF	Disable Receive End of Frame Select: When this bit is set to 1, RX EOF interrupts are disabled. Default value is 0.
0	RX EOC†	Disable Receive End of Channel Select: When this bit is set to 1, the receive channel of RX EOC interrupts are disabled. Default value is 0.

† Refer to Chapter 5 List Structures to determine how transmit and receive channels can be checked to insure they are disabled.

A.4 10Base-T PHY Registers

The 10Base-T PHY registers are indirectly accessible through the MII. This is a low-speed serial interface which is supported on ThunderLAN through the NetSio register in adapter DIO space. A host software program uses the MCLK, MTXEN, and MDATA bits in this register to implement the MII serial protocol for the management interface.

The 802.3u MII serial protocol allows for up to 32 different PMDs, with up to 32 (16-bit wide) internal registers in each device. The 10Base-T PHY implements seven internal registers, three of which are hardwired. The diagram below shows the devices register map. The registers shown in gray are the generic registers mandated by the MII specification. The registers shown in white are TI-specific registers.

Figure A–8. 10Base-T PHY Registers

	Register	Description
GEN_ctl	0x00	PHY generic control register
GEN_sts	0x01	PHY generic status register
GEN_id_hi	0x02	PHY generic identifier (high)
GEN_id_lo	0x03	PHY generic identifier (low)
AN advertisement	0x04	Autonegotiation advertisement
AN link-partner ability	0x05	Autonegotiation link-partner ability
AN expansion	0x06	Autonegotiation expansion
Reserved	0x07	
Reserved	through	Reserved by 802.3
Reserved	0x0F	
TLPHY_id	0x10	ThunderLAN PHY identifier
TLPHY_ctl	0x11	ThunderLAN PHY control register
TLPHY_sts	0x12	ThunderLAN PHY status register

A.4.1 PHY Generic Control Register – GEN_ctl @ 0x0

Byte 1 Byte 0

15	14	13	12	11	10	9	8	7	6 5 4 3 2 1 0
RESET	LOOPBK	0	AUTO ENB	PDOWN	ISOLATE	AUTO RSRT	DUPLEX	COL TEST	Reserved

Table A–19. PHY Generic Control Register Bits

Bit	Name	Function
15	RESET	PHY reset: Writing a 1 to this bit causes the PHY to be reset. This bit is self-clearing. The bit returns a value of 1 when read until the internal reset is complete.
14	LOOPBK	Loopback: This bit enables/disables internal loopback within the PHY device. When this bit is set to a 1 (default), data is internally wrapped within the PHY and does not appear on the network. When this bit is set to 0, data is transmitted to and received from the network. While the PHY is in the loopback state, all network lines are placed in a noncontentious state.
13	0	Speed selection bit: Not implemented
12	AUTOENB	Autonegotiation enable: This bit enables/disables the autonegotiation process. If this bit is clear, the link shall be configured via the DUPLEX bit and the PHY will implement the standard 10Base-T link integrity test. The default value of this bit is enabled. If this bit is set to one, the PHY engages in autonegotiation when a link-fail condition is detected. The link will not be valid until the AUTOCMPLT bit is set to one. The PHY does not automatically configure itself after autonegotiation has completed. Driver software must determine from the contents of the AN_adv, AN_lpa, and AN_exp registers what the correct setting for DUPLEX should be, or whether the link partner does not implement 10Base-T.
11	PDOWN	Power-down: When this bit is set (default), the PHY is placed in a low-power consumption state. The time required for the PHY to power up after this bit is cleared can vary considerably, primarily based on whether a crystal or crystal oscillator is connected to FXTL1/FXTL2 (around 50 ms for the former, and less than 2 ms for the latter). It is good practice to set the RESET bit after this time to ensure the PHY is in a valid state (this is not necessary when a crystal oscillator is used).

Table A–19. PHY Generic Control Register Bits (Continued)

Bit	Name	Function
10	ISOLATE	Isolate: When this bit is set (default), the PHY electrically isolates its data paths from the MII. In this state, it does not respond to the MTXD0–3, MTXEN, and MTXER pin inputs, and presents a high impedance on its MTCLK, MRCLK, MRXDV, MRXER, MRXD0–3, and MCOL pin outputs. It, however, still responds to management frames on the MDIO and MDC pins. Due the the embedded nature of the PHY, the isolate function has no visible effect on the MII pin interface.
9	AUTORSRT	Restart autonegotiation: The autonegotiation process is restarted by setting this bit to 1. This bit is self-clearing, and the PHY returns a value of 1 in this bit until the autonegotiation process is initiated.
8	DUPLEX	Duplex mode: Setting this bit to a 1 configures the PHY for full-duplex 10Base-T operation, whereas setting this bit to 0 (default) configures the PHY for half-duplex operation. In AUI mode, the PHY is capable of full-duplex operation. The mode is determined by the external device to which the PHY is interfaced, rather than this bit (which has no effect on PHY operation).
7	COLTEST	Collision test mode: Setting this bit to 1 causes the PHY to assert the collision sense signal MCOL whenever the transmit enable MTXEN pin is asserted.
6–0	Reserved	Read as 0

A.4.2 PHY Generic Status Register – GEN_sts @ 0x1

Byte 1 Byte 0

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	1	Reserved					AUTOCOMPLT	RFLT	1	LINK	JABBER	1

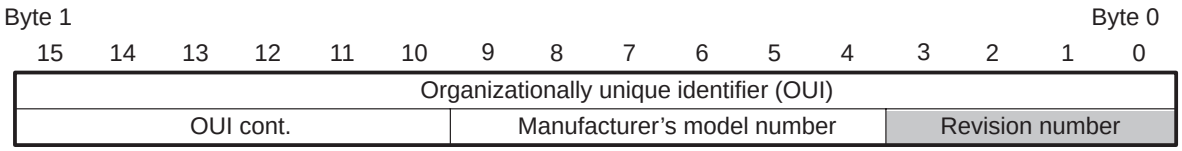
Table A–20. PHY Generic Status Register Bits

Bit	Name	Function
15	0	100Base-T4 capable: Not supported
14	0	100Base-Tx full-duplex capable: Not supported
13	0	100Base-Tx half-duplex capable: Not supported
12	1	10Base-T full-duplex capable: This bit is hardwired to 1 to indicate that 10Base-T full duplex is supported.
11	1	10Base-T half-duplex capable: This bit is hardwired to 1 to indicate 10Base-T half duplex is supported.
10–6	Reserved	Read as 0
5	AUTOCMPLT	Autoconfiguration complete: When this bit is set, it indicates that the autonegotiation process has finished or was not enabled. If autonegotiation is enabled, this bit also indicates that the contents of registers AN_adv, AN_lpa and AN_exp are valid. This bit is 0 only during an actual negotiation process.
4	RFLT	Remote fault: This bit mirrors the LPRFLT bit received in the most recent autonegotiation link code word. When set to 1, this bit indicates that the link partner is in a fault condition.
3	1	Autonegotiation ability: The PHY supports autonegotiation.
2	LINK	Link status: When this bit is read as 1, it indicates that the PHY has determined that a valid 10Base-T link has been established. When read as 0, it indicates that the link is not valid. A link invalid state is latched (held) until the register is read. This bit has no meaning if the AUI interface is selected. The PHY implements the standard 10Base-T link integrity test state machine. Link pulses are expected to be seen every 8–24 ms to maintain a good link. If no link pulses are seen for over 100 ms, the link-fail state is entered and this bit is cleared. If AUTOENB is not set, the bit is set again after seven consecutive, correctly timed link pulses are received. If AUTOENB is set, the link fail causes the autonegotiation process to restart.

Table A–20. PHY Generic Status Register Bits (Continued)

Bit	Name	Function
1	JABBER	Jabber detect: When read as 1 this bit indicates a 10Base-T jabber condition has been detected. A jabber condition is latched (held) until the register is read. This bit has no meaning if the AUI interface is selected. The jabber condition occurs when a single packet transmission exceeds 20 ms (this cannot happen through normal TLAN operation). In the jabber condition, all transmit requests are ignored and collision detection is disabled, as is the internal loopback of transmit data (when in half-duplex mode). The jabber condition persists for 28–576 ms after the deassertion of the MTXEN pin.
0	1	Extended capability: This bit is hardwired to 1 to indicate that the extended register set is supported.

A.4.3 PHY Generic Identifier – GEN_id_hi/GEN_id_lo @ 0x2/0x3



These two hardwired 16-bit registers contain an identifier code for the TLAN 10Base-T PHY. GEN_id_hi contains 0x4000, GEN_id_lo contains 0x50xx, where the xx denotes the revision.

The revision number value - xx has changed from 14 to 15 for the GEN_id_lo register.

A.4.4 Autonegotiation Advertisement Register – AN_adv @ 0x4

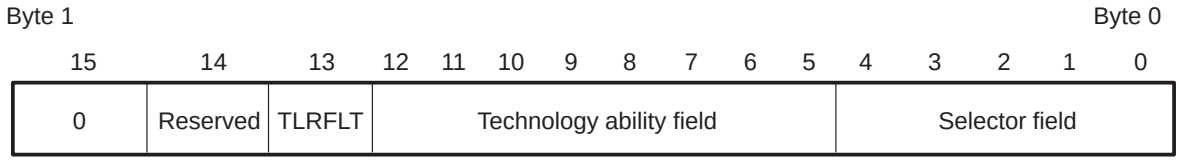


Table A–21. Autonegotiation Advertisement Register Bits

Bit	Name	Function
15	0	Autonegotiation next page: Reception/transmission of autonegotiation next pages is optional and not supported by this PHY.
14	Reserved	For internal use of the autonegotiation process.
13	TLRFLT	TLAN remote fault: This bit enables TLAN to indicate a remote fault condition to the link partner.
12–5	Technology ability field	Autonegotiation advertised technology ability: This 8-bit value is sent to the link-partner to indicate the abilities of the TLAN PHY. Unsupported abilities cannot be advertised, which for this PHY means only two bits have meaning when set: ❏ Bit 6: PHY supports full-duplex 10Base-T ❏ Bit 5: PHY supports half-duplex 10Base-T ❏ Bits 12 through 7 are read only, and set to 0.
4–0	Selector field	Autonegotiation selector field code: This field has a hardwired value of 0001, meaning the PHY only supports 802.3 format link code words.

A.4.5 Autonegotiation Link Partner Ability Register – AN_lpa @ 0x5

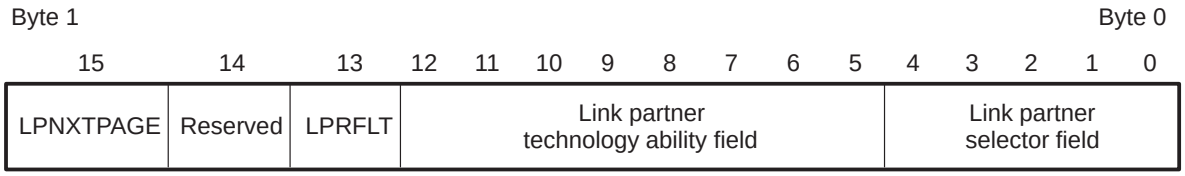


Table A–22. Autonegotiation Link Partner Ability Register Bits

Bit	Name	Function
15	LPNXTPAGE	Link partner next page: When this bit, is set, the link partner indicates that it has another page to send. Reception of autonegotiation next pages is optional and is not supported by this PHY.
14	Reserved	For internal use of the autonegotiation process
13	LPRFLT	Link partner remote fault: When this bit is set to a 1, the link partner reports a remote fault condition.
12–10		Reserved: For future IEEE-defined abilities
9		100Base-T4: Set to 1 if supported by the link partner
8		100Base-Tx full duplex: Set to 1 if supported by the link partner
7		100Base-Tx: Set to 1 if supported by the link partner
6		10Base-T full duplex: Set to 1 if supported by the link partner
5		10Base-T: Set to 1 if supported by the link partner
4–0	Link partner selector field	This five-bit field encodes the format of this register. ThunderLAN only supports IEEE 802.3 format fields (as detailed in bits 12 through 5 above), code 00001. (The only other currently specified value is 00010, for 802.9a multimedia frames).

A.4.6 Autonegotiation Expansion Register – AN_exp @ 0x6

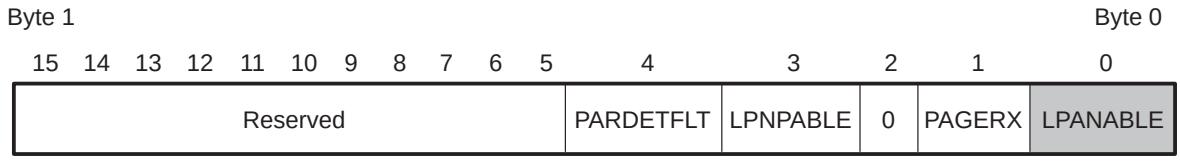


Table A–23. Autonegotiation Expansion Register Bits

Bit	Name	Function
15–5	Reserved	Read as 0
4	PARDETFLT	Parallel detection fault: For multi-technology PHYs, this bit indicates multiple valid links. This PHY only supports a single technology (10Base-T) and so this bit should be ignored.
3	LPNPABLE	Link partner next page able: When this bit is set to 1, the link partner indicates that it implements autonegotiation next page abilities.
2	0	Next page able: ThunderLAN does not support next page transmission or reception.
1	PAGERX	Page received: This bit is set after three identical and consecutive link code words have been received from the link partner and the link partner has indicated that it has received three identical and consecutive link code words from ThunderLAN. This bit is cleared when read.
0	LPANABLE	Link-Partner Autonegotiation Able: When this bit is set to a one, the PHY is receiving autonegotiation fast link pulse bursts from the link partner. This bit is reset to zero if the Link-Partner is not autonegotiation able.

A.4.7 ThunderLAN PHY Identifier High/Low – TLPHY_id @ 0x10

This hardwired 16-bit register contains a TI assigned identifier code for the ThunderLAN PHY/PMIs. An additional identifier is required to identify non-802.3 PHY/PMIs, which are not otherwise supported by the 802.3u MII specification. The identifier code for the internal 10Base-T/AUI PHY is 0x0001.

A.4.8 ThunderLAN PHY Control Register–TLPHY_ctl @ 0x11

Byte 1 Byte 0

15	14	13	12	11	10 9 8 7 6 5 4 3	2	1	0
IGLINK	SWAPOL	AUISEL	SQEEN	MTEST	Reserved	NFEW	INTEN	TINT

Table A–24. ThunderLAN PHY Control Register Bits

Bit	Name	Function
15	IGLINK	Ignore link: When this bit is set to 0, the 10Base-T PHY expects to receive link pulses from the link partner (hub, switch, etc.), and sets the LINK bit in the GEN_sts register to 0 if they are not present. When this bit is set to 1, the internal link integrity test state machine is forced to stay in the link-good state, even when no link pulses are received. The LINK bit is set to 1.
14	SWAPOL	Swap polarity: Writing a 1 to this bit causes the PHY to reverse the polarity of the 10Base-T receiver input pair. This is used to compensate for a cable in which the receive pair has been incorrectly wired.
13	AUISEL	AUI select: Writing a 1 to this bit causes the PHY to use the AUI network interface; writing 0 (default) causes the PHY to use the 10Base-T network interface. The transmitters and receivers on the PHY are multiplexed between AUI and 10Base-T, so both cannot operate simultaneously.
12	SQEEN	SQE (signal quality error) enable: Writing a 1 to this bit causes the 10Base-T PHY to perform the SQE test function at the end of packet transmission. The SQE function is only performed in 10Base-T mode. The SQE test provides an internal simulated collision to test the collision detect circuitry. It asserts the MCOL bit between 600–1600 ns after the last positive edge of a frame is transmitted, with the collision lasting between 500 and 1500 ns.
11	MTEST	Manufacturing test: When this bit is set to a 1, the PHY is placed into manufacturing test mode. Manufacturing test mode is reserved for Texas Instruments manufacturing test only. Operation of the PHY and this register is undefined when this bit is set.
10–3	Reserved	Read and write as 0
2	NFEW	Not far end wrap: This bit only has meaning when the LOOPBK bit of the GEN_ctl is a 1. Writing a 1 to this bit causes the PHY to wrap the Tx data to the Rx data at the MII. Writing a 0 to this bit causes the PMI to wrap the Tx data to the Rx data at the furthest point possible. When NFEW is set to a 1, the preamble wraps without degradation; when it is set to a 0, the PHY needs only to wrap back the start of frame delimiter (SFD).

Table A–24. ThunderLAN PHY Control Register Bits (Continued)

Bit	Name	Function
1	INTEN	Interrupt enable: Writing a 1 to this bit allows the PHY to generate interrupts on the MII if the MINT bit is set. Writing a 0 to this bit prevents the PHY from generating any MII interrupts. This bit does not disable test interrupts.
0	TINT	Test interrupt: Writing a 1 to this bit causes the PHY to generate an interrupt on the MII. Writing a 0 to this bit causes the PHY to stop generating an interrupt on the MII. This test function is totally independent of the INTEN and MINT bits. This bit is used for diagnostic testing of the MII interrupt function.

A.4.9 ThunderLAN PHY Status Register – TLPHY_sts @ 0x12

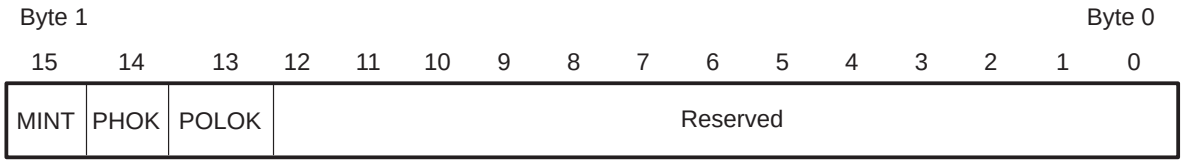


Table A–25. ThunderLAN PHY Status Register Bits

Bit	Name	Function
15	MINT	MII interrupt: This bit indicates an MII interrupt condition. The MII interrupt request is activated (and latched) until the register is read. Writing to this bit has no effect. This bit is set to a 1 when: ❑ PHOK is set to 1 ❑ LINK changes state or is different from either the last read value or the current state of the link ❑ RFLT is set to 1 ❑ JABBER is set to 1 ❑ PLOK is set to 1 ❑ PAGERX is set to 1 ❑ AUTOCMPLT is set to 1 ❑ TPENERGY is set to 1
14	PHOK	Power high OK: This bit indicates that the internal crystal oscillator circuit has performed 75 oscillations (cycles). PHY-sourced clocks (MRCLK and MTCLK) are not valid until this bit is asserted. If a crystal is connected to FXTL1/FXTL2 rather than a crystal oscillator, the clocks may take up to 50 ms to become stable and the PHY requires the RESET bit be set to ensure it is in a valid state. When the state of this bit changes, the PSTATE bit in the TLPHY_sts register is set.

Table A–25. ThunderLAN PHY Status Register Bits (Continued)

Bit	Name	Function
13	POLOK†	Polarity OK: When this bit is high (default), the 10Base-T PHY receives valid (noninverted) link pulses. If this bit goes low, it indicates that a sequence of seven consecutive inverted link pulses has been detected.
12	TPENERGY	Twisted-pair energy detect: This bit only has meaning in AUI mode. When set to a 1, it indicates that the PHY receives of impulses on the FRCVP/FRCVN pins. Energy is detected in the form of link pulses, the sense of which is determined by the POLREV bit. Note that if the POLREV bit is set incorrectly, link pulses are not seen and this bit is not set.†
11–0	Reserved	Read as 0

† This function is provisional. It is possible for a link pulse with gross undershoot to appear as an inverted link pulse.

Appendix B

TNETE211 100VG-AnyLAN Demand Priority Physical Media Independent (PMI) Interface

This appendix contains register definitions for the TNETE211 100VG-AnyLAN PMI interface. ThunderLAN uses these registers to store information on its internal status and its communication with the host. This appendix describes the purpose and function of each register and provides many bitmaps and descriptions of individual bits. The appendix also describes the sequence of steps used for IEEE 802.12 100VG-AnyLAN training, which is used in opening ThunderLAN to the network.

Topic **Page**

B.1 100VG-AnyLAN Training

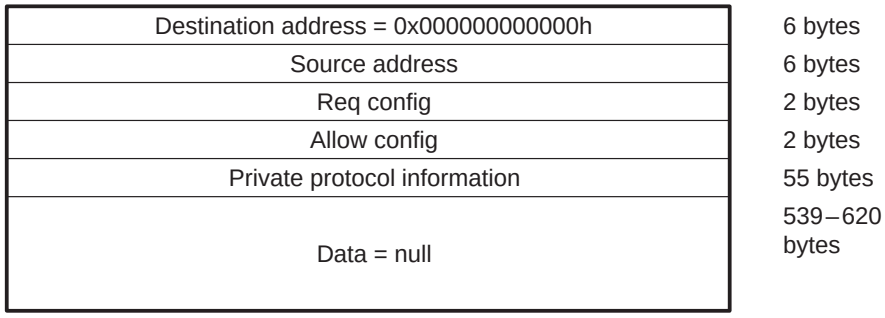
The algorithm used to open ThunderLAN to the network depends on the network protocol in use. The demand priority protocol specified in IEEE 802.12 goes through a training process to open onto the wire. To open the controller the driver must:

- Enter VG training; the network protocol is demand priority.
- Issue a dummy interrupt by asserting Req_Int in the HOST_CMD register. Wait one second for this interrupt to process. If the interrupt is handled, then ThunderLAN is open on the wire. Otherwise, it is not.

Training between the client and hub is accomplished by exchanging 24 consecutive frames (training frames) between the client and hub. These 24 frames must be exchanged within a window consisting of 48 frames. If training is not accomplished within this window, it can continue after a suitable delay in another 48-frame window.

The following shows the format of an 802.12 training frame:

Figure B–1. 802.12 Training Frame Format



The following describes what the driver must do to successfully train:

1) Assert the INTEN bit in the TLPHY_ctl register to enable MII interrupts to ThunderLAN from the voice grade (VG) PHY

2) Ensure that ThunderLAN is not in copy all frames mode, copy short frames mode, or broadcast mode (CAF, CSF, and NOBRX bits in the NetCmd register)

3) Disable the multicast addresses contained in the HASH registers

4) Set the general purpose address register AREG0 to 0x000000000000h so that ThunderLAN may receive the training frames. Note that the training frame contains a destination address of 0.

5) Determine if ThunderLAN is currently transmitting as it enters the training procedure. If so, stop the transmitter.

   a) If transmitting:

      i) Set up a deadman timer before halting the transmitter (10 s)

      ii) Assert the Tx STOP interrupt by asserting the MASK5 bit in the NetMask register

      iii) Issue a Tx STOP command by setting the STOP bit in the HOST_CMD register with the appropriate channel set

      iv) Exit the routine and the service interrupt when the transmitter stops

   b) If not transmitting, do not use STOP

6) At this point, use the following steps to build the training frame in a buffer:

   - Read all the statistics registers to clear them
   - Move the pointer to the training frame buffer
   - Set the destination address to NULL as specified in 802.12
   - Set the source address to your adapter's source address
   - Set the Req config field to the appropriate options
   - Fill in the rest of the training frame (two bytes in the Allow config field + 675 bytes of data) with NULLs
   - Setup the Tx list with the training buffer's pointer and size in preparation for a Tx GO command

7) Request the beginning of the training period by clearing the TRFAIL bit and asserting the TRIDL bit in the TLPHY_ctl register.

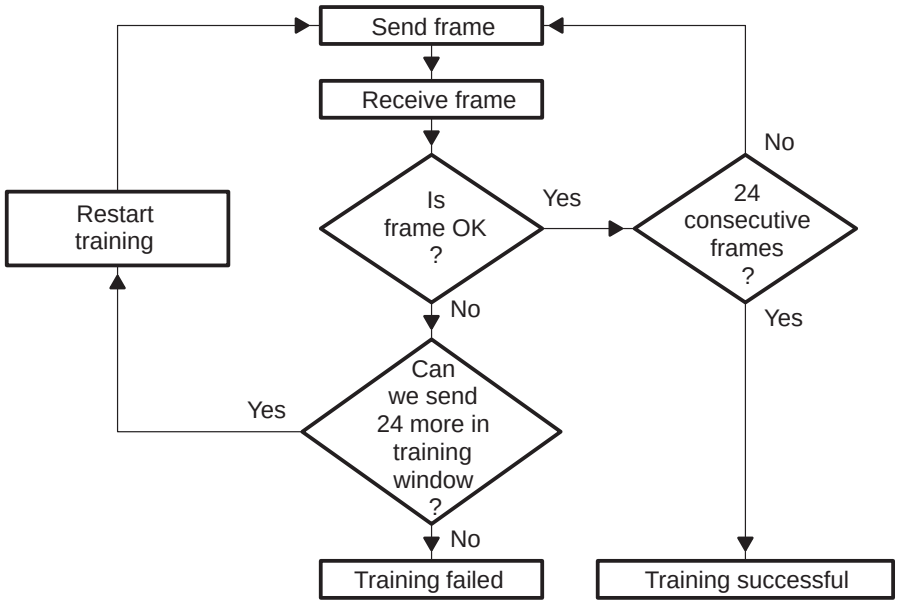
8) The driver now waits for a status interrupt. The MASK7 bit in the NetMask register must be set for the status interrupt to reach ThunderLAN.

9) When this interrupt arrives, perform frame exchange

Training involves the exchange of 24 consecutive training frames between the client and the hub. The client begins by sending a training frame. The hub answers with the same frame, except in the Allow config field. The client verifies that the received frame is a valid training frame. This process continues until 24 successful consecutive frames are exchanged.

The 802.12 standard states that these 24 frames must be exchanged within a training window of 48 frames. If this fails, training may or may not be possible within this window. The client must ensure that there are enough frames left in the window for successful training. If this is not possible, the client must request a new training window and try again. The process is shown below:

Figure B–2. Training Flowchart



To determine whether the training frame received from the hub is correct the driver should:

- ❑ Check that the destination address is a null
- ❑ Check that the training frame has the adapter's address for the source address. This is not required, but is good practice.
- ❑ Check the first 55 bytes of the data field (optional private protocol information area as defined in 802.12). They should be null.

If the training frame passes these criteria, it is valid. The driver updates a counter showing the number of consecutive valid training frames passed. The driver also keeps a separate counter showing how many frames are left in the training window.

If the training frame does not pass the criteria, it is invalid. The driver must use the counter which shows how many frames are left in the training window, and if it is equal to or greater than 24, it restarts the exchange of frames.

If the training window does not allow the exchange of 24 frames, the driver must request a new training window. This can be accomplished by:

- Setting the TRFAIL bit and clearing the TRIDLE bit in the TLPHY_ctl register
- Waiting for a status interrupt. A deadman timer (10 s) may be necessary to ensure that the driver does not sit indefinitely in this state
- Checking the RETRAIN bit in TLPHY_sts when status interrupt arrives
- Requesting the beginning of the training period by clearing the TRFAIL bit and asserting the TRIDL bit in the TLPHY_ctl register

If the driver has successfully trained, the driver clears the TRIDLE bit in TLPHY_ctl and exits the training routine. You must reinitialize the AREG0 register to this adapter's address, the HASH registers, and the CAF, CSF, and NOBRX in the NetCmd register to their chosen values.

B.2 TNETE211 Register Descriptions

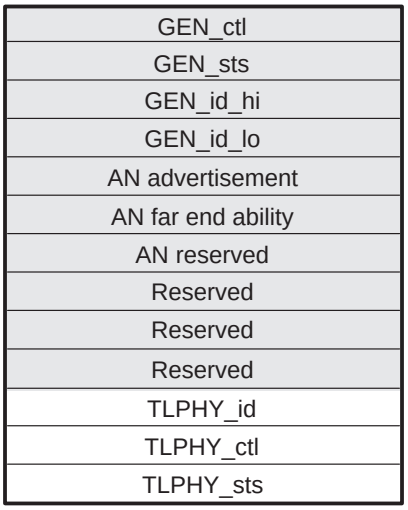
This document is a specification for ThunderLAN's TNETE211 PMI device, which interfaces the ThunderLAN MII and the PMD device. It is responsible for converting the nibble stream of data from the MII to the four-pair category-3 cabling, and from the four-pair category-3 cabling to the MII.

The TNETE211 connects to ThunderLAN's IEEE 802.3u-compliant MII and converts the data and control signals into a fully compliant 802.12 MII for 100VG-AnyLAN operation. The TNETE211 is responsible for implementing much of the 100VG's functionality, including data channeling, ciphering/deciphering, and encoding/decoding. It also implements the 802.12 media access controller (MAC) state machines.

The 100VG-AnyLAN demand priority PHY registers are indirectly accessible through the MII management interface present in ThunderLAN. This is a low-speed serial interface which is supported on ThunderLAN through the NetSio register in adapter DIO space. A host software program uses the MCLK, MTXEN, and MDATA bits in this register to implement the MII serial protocol for the management interface.

The 802.3u MII serial protocol allows for up to 32 different PMDs, with up to 32 (16 bit wide) internal registers in each device. The 100VG-AnyLAN demand priority PHY implements seven internal registers, three of which are hard-wired. The diagram below shows the devices' register map. The registers shown in gray are the generic registers as mandated by 802.3u. The registers shown in white are Texas Instruments specific registers. All other registers are read as 0s.

Figure B–3. TNETE211 Registers



Register	Description
0x00	PHY generic control register
0x01	PHY generic status register
0x02	PHY generic identifier (high)
0x03	PHY generic identifier (low)
0x04	Not implemented
0x05	Not implemented
0x06	Not implemented
0x07 through 0x0F	Reserved by 802.3
0x10	ThunderLAN PHY identifier
0x11	ThunderLAN PHY control register
0x12	ThunderLAN PHY status register

B.2.1 PHY Generic Control Register–GEN_ctl @ 0x0

Byte 1 Byte 0

15	14	13	12	11	10	9	8	7	6 5 4 3 2 1 0
RESET	LOOPBK	0	0	PDOWN	ISOLATE	0	0	COL TEST	Reserved

Table B–1. PHY Generic Control Register Bits

Bit	Name	Function
15	RESET	PHY reset: Writing a 1 to this bit causes the PHY and its internal registers to reset. This bit is self-clearing; the bit returns a value of 1 when read until the internal reset is complete. This bit also serves to reset the 802.12 MAC state machine to MAC0.
14	LOOPBK	Loopback: This bit enables/disables internal loopback within the PHY device. When this bit is set to 1 (default), data is internally wrapped within the PHY and does not appear on the network. When this bit is set to 0, data is transmitted to and received from the network. While the PHY is in the loopback state, all network lines are placed in a noncontentious state. This bit also resets the 802.12 MAC state machine to MAC0.
13	0	Speed selection bit: Not implemented
12	0	Autoconfiguration enable: Not implemented

Table B–1. PHY Generic Control Register Bits (Continued)

Bit	Name	Function
11	PDOWN	Power down: When this bit is set (default), the PHY is placed in a low-power consumption state. This bit resets the 802.12 MAC state machine to MAC0. It stops the Tx and Rx functions and disables the oscillator by deasserting POSCEN†. In power-down mode, PDOWN is the only bit that can be written to.
10	ISOLATE	Isolate: When this bit is set (default), the PHY electrically isolates its data paths from the MII. In this state it does not respond to MTKD[3::0], MTXEN, and MTXER inputs, and presents a high impedance on its MTCLK, MRCLK, MRXDV, MRXER, MRXD0–3, and MCOL outputs. It will, however, still respond to management frames on MDIO and MDCLK.
9	0	Autoconfiguration enable: Not implemented
8	0	Duplex mode: VG currently does not support a full-duplex mode
7	COLTEST	Collision test mode: Setting this bit to 1 causes the PHY to assert the collision sense signal MCOL whenever transmit enable (MTXEN) is asserted.
6–0	Reserved	Read as 0

† Oscillator enable from the TNETE211

B.2.2 PHY Generic Status Register – GEN_sts @ 0x1

Byte 1 Byte 0

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	Reserved					0	RFLT	0	LINK	JABBER	1

Table B–2. PHY Generic Status Register Bits

Bit	Name	Function
15	0	100Base-T4 capable: Not supported
14	0	100Base-Tx full-duplex capable: Not supported
13	0	100Base-Tx half-duplex capable: Not supported
12	0	10Base-T full-duplex capable: Not supported
11	0	10Base-T half-duplex capable: Not supported
10–6	Reserved	Read as 0
5	0	Autoconfiguration complete: Not implemented

Table B–2. PHY Generic Status Register Bits (Continued)

Bit	Name	Function
4	RFLT	Remote fault: When this bit is set, it indicates that a remote fault condition has been detected. This bit is autoclearing, and a remote fault condition is latched (held) until the register is read.
3	0	Autoconfiguration capable: Not supported
2	LINK	Link status: When this bit is read as 1, it indicates that the PHY has determined that a valid link has been established. When read as 0, it indicates that the link is not valid. This bit is autoclearing, and a link invalid state is latched (even if the link returns) until the register is read. This is to ensure that the driver can determine the type of MII interrupt.
1	JABBER	Jabber detect: When read as 1, this bit indicates a jabber condition has been detected. This bit is autoclearing, and a jabber condition is latched (held) until the register is read.
0	1	Extended capability: This bit is hardwired to 1 to indicate that the extended register set is supported.

B.2.3 PHY Generic Identifier – GEN_id_hi/GEN_id_lo @ 0x2/0x3

These two hardwired 16-bit registers contain an identifier code for the ThunderLAN 100VG-AnyLAN demand priority PHY. The actual value is 0x4000/ 0x502x for this PMI device, where x denotes the PHY revision.

B.2.4 ThunderLAN PHY Identifier High/Low – TLPHY_id @ 0x10

This hardwired 16-bit register contains a Texas Instruments-assigned identifier code for the ThunderLAN PHY/PMIs. An additional identifier is required to identify non-802.3 PHY/PMIs, which are not otherwise supported by the 802.3u MII specification. The value for the 100VG-AnyLAN demand priority is 0x0002.

B.2.5 ThunderLAN PHY Control Register – TLPHY_ctl @ 0x11

Byte 1 Byte 0

15	14	13	12	11 10 9 8 7 6	5	4	3	2	1	0
IGLINK	MCRS	PTLSWEN	PRLSREN	Reserved	TRFAIL	TRIDLE	NPMDW	NFEW	INTEN	TINT

Table B–3. ThunderLAN PHY Control Register Bits

Bit	Name	Function
15	IGLINK	Ignore link: When this bit is set to 0, the 100VG-AnyLAN Demand Priority PHY expects to receive link pulses from the hub, and sets the LINK bit in the GEN_sts register to 0 if they are not present. When this bit is set to 1, link pulses are ignored and the LINK bit is always set to 1.
14	MCRS	MCRS output value: The MCRS pin of the PMI is deasserted when the transmit/receive medium is idle. Once the transmit/receive medium is nonidle, the pin is asserted.
13	PTLSWEN	PMD TLS write control value: The PTLSWEN pin of the PMI is used to control the PMD TLS write control. It should be set to 0 for all normal operations.
12	PRLSREN	PMD RLS read control value: The PRLSREN pin of the PMI is used to control the PMD RLS read control. It should be set to 0 for all normal operations.
11–6	Reserved	Read as 0s
5	TRFAIL	Training fail indicator: Writing a 1 to this bit causes the PMI to restart training when the next window is reached. This bit forces the PMI to interrupt the driver with a retrain event when retraining occurs.
4	TRIDLE	Training idle request: Writing a 1 to this bit causes the PMI to indicate training idle to the PMD whenever there is no transmit request pending. Writing a 0 to this bit causes the PMI to send idle up whenever there is no transmit request pending.
3	NPMDW	Not physical media dependant wrap: This bit only has meaning when the LOOPBK bit of the GEN_ctl is a 1. Writing a 1 to this bit causes the PMI to wrap the Tx data to the Rx data at the far side of the PMI. Writing a 0 to this bit causes the PMI to wrap the Tx data to the Rx data in the analog device attached to the PMI.
2	NFEW	Not far end wrap: This bit only has meaning when the LOOPBK bit of the GEN_ctl is a 1. Writing a 1 to this bit causes the PMI to wrap the Tx data to the Rx data at the MII interface. Writing a 0 to this bit causes the PMI to wrap the Tx data to the Rx data based on the value of the NPMDW bit.
1	INTEN	Interrupt enable: Writing a 1 to this bit causes the PMI to generate interrupts to the MII if any one of the event conditions occur. Writing a 0 to this bit causes the PMI to not generate an MII interrupt even though an event condition has occurred.
0	TINT	Test interrupt: Writing a 1 to this bit causes the PMI to generate an interrupt to the MII. Writing a 0 to this bit causes the PMI to not generate an MII interrupt. This bit is used to test the interrupts from the PHY prior to requiring them.

B.2.6 ThunderLAN PHY Status Register–TLPHY_sts @ 0x12

Byte 1 Byte 0

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MINT	PHOK	0	0	CONFIG				0	RETRAIN	LSTATE		TRFRTO	RTRIDL	LRCV	LSIL

Table B–4. ThunderLAN PHY Status Register Bits

Bit	Name	Function
15	MINT	MII interrupt: This bit indicates an MII interrupt condition. The MII interrupt request is activated whenever this bit is set to 1. The bit may be cleared and the interrupt deasserted by writing a 1 to this bit position. Writing a 0 to this bit has no effect. This bit is set to 1 when: ❑ The state of the PHOK, LINK, JABBER, RETRAIN TRFRTO, RTRIDL, or LSIL changes ❑ RFLT is set to 1 The condition for the interrupt can be determined by examining these bits. In the case of LINK, which could change before the interrupt type can be determined, ThunderLAN keeps this bit as 0 until it is read, even if the link is restored. This is to ensure that the condition change is not lost.
14	PHOK	Power high OK: This bit is reserved for future use.
13–12	Reserved	Read as 0s
11–8	CONFIG	Configuration bits: These bits indicate the type of PMD attached to the PMI device.
7	Reserved	Read as 0
6	RETRAIN	Retrain link: This bit, when set, indicates that the link must be retrained. It is set if the link has been silent too long, bad RLS codes have been detected, training idles have been received from the hub, or Rx or Tx jabber has been detected. If the INTEN bit is also set, this causes an MII interrupt.
5–4	LSTATE	Link state: This field returns the PMD current link state bits. It is used for informational purposes only. ❑ 00: Link is silent ❑ 01: Data on link ❑ 10: Control on link ❑ 11: Bad RLS code on link

Table B–4. ThunderLAN PHY Status Register Bits (Continued)

Bit	Name	Function
3	TRFRTO	Training frame time out: This bit indicates that the PMI is in training, the training frame has not been received in 273 μs, and that another training frame should be sent. If the INTEN bit is also set, this causes an MII interrupt.
2	RTRIDL	Receive training idles: This bit indicates that the the PMI is receiving training idles from the repeater, indicating that the station should enter training. This bit is autoclearing, and is latched until read. If the INTEN bit is also set, this causes an MII interrupt.
1	LRCV	Long receive: This bit indicates that the PMI has detected a long receive (receive jabber) condition from the PMA. This bit is autoclearing, and is latched until read. If the INTEN bit is also set, this causes an MII interrupt.
0	LSIL	Long silence: This bit indicates that the PMI has detected a long silence condition from the PMA. This bit is autoclearing and is latched until read. If the INTEN bit is also set, this causes an MII interrupt.

Appendix C

TNETE100PM/TNETE110PM

For information on the TNETE100PM and TNETE110PM implementations of ThunderLAN, please contact TLANHOT@micro.ti.com, which is listed on page v of this document.