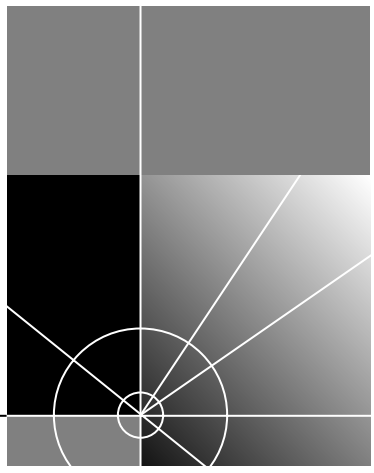
3C359 Network Interface Card Technical Reference

TokenLink Velocity® XL PCI token ring NIC with Parallel Tasking® II technology



http://www.3com.com/

Part Number: 89-0765-000
Published August 1998

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

CONTENTS

4 CONFIGURATION

5 EEPROM

6 DOWNLOAD AND TRANSMISSION

7 RECEPTION AND UPLOAD

10 OTHER REGISTERS

11 SOFTWARE OPERATION

FIGURES

TABLES

1 INTRODUCTION

This technical reference describes the basic architecture and defines the programming interface of the 3Com® TokenLink Velocity® XL PCI token ring network interface card (3C359 NIC). See Table 1.

Table 1 3Com 3C359 NIC Summary

Model	Bus	Ring Speed (Mbps)	Cable	Connector
3C359	PCI	16 (default) or 4	Two-pair Category 3, 4, or 5 UTP; or Type 1 or 6 STP	RJ-45 or DB-9

NIC Features

3C359 NICs have these features:

- Support for 4 Mbps and 16 Mbps IEEE 802.5 and IBM-compatible token ring LANs
- Support for IEEE P802.5r Draft 2 (dedicated token ring)
- Fully independent transmit and receive data paths for full-duplex operation
- Support for both promiscuous and receive all group/multicast packet modes
- Built-in serial EEPROM controller for configuration and network address storage
- Support for Remote Wake-Up
- Support for the maximum size packet (~18 KB)
- Parallel Tasking® II technology for superior performance
- Plug and play configurability
- Hash filtering of multicast packets
- Optional flash ROM for BIOS code
- Bus-mastering PCI for ultra-low CPU utilization
- Multipacket, multifragment DMA scatter and gather operations for uploads and downloads
- 64 KB SRAM for microcode and data buffers

About This Technical Reference

This technical reference contains programming interface information that software engineers, independent software developers, and test engineers need to write device drivers, diagnostic programs, and production test software for 3C359 NICs. This information includes:

- Theory of operation; for example, how transmission and reception occur.

- Register set, including the size, type, address, and function of each register and the functions of the bits in the register.
- Software interface, which allows communication between the NIC and the host.

The information in this reference is language-independent. It applies regardless of the programming language you use to write the driver or other software program.

In this reference, addresses refer to physical addresses, not to logical or virtual addresses. Numeric values other than decimal values are presented in the following formats:

Format	Description	Example
#'rZZZZ	# is the number of bits.	6'b100101 is a 6-bit binary notation.
	' is a delimiter.	6'h25 is a 6-bit hexadecimal notation.
	r is the radix (b for binary and h for hexadecimal).	
	ZZZZ is the value.	
ZZZr	ZZZ is the value.	100101b is a binary notation.
	r is the radix (b for binary and h for hexadecimal).	25h is a hexadecimal notation.

Terms and Acronyms

The following terms and acronyms are used in this reference:

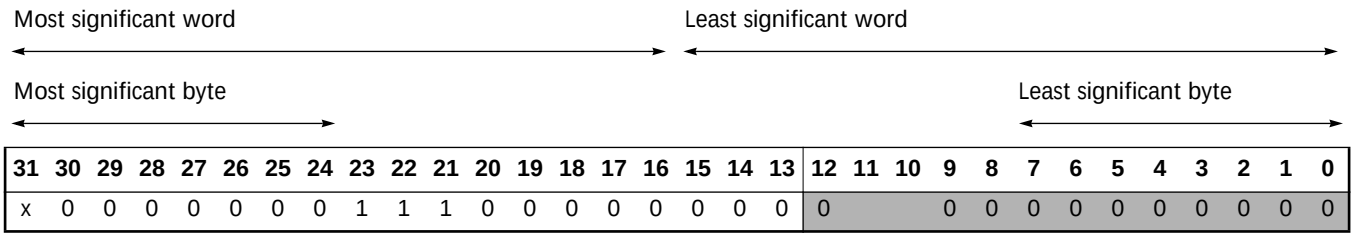
Term or Acronym	Meaning
ACA	Attachment control area, the base address for MMIO registers.
ARB	Adapter request block, a control block in shared RAM that passes receive information or commands from the CP to the host.
ASB	Adapter status block, a control block in shared RAM that passes host responses to ARB commands to the CP.
BIOS	Basic input/output system. BIOS code is executed when the system boots.
BIOS ROM	Read-only memory (ROM) that contains code that is executed when the system boots and performs remote program load over the network.
BIST	Built-in self-test.
byte	An 8-bit wide quantity of data.
CP	Communications processor (V30H).
CRC	Cyclic redundancy check.
DA	Destination address.
double word (dword)	A 32-bit wide quantity of data (4 bytes).
download	The process of transferring transmit data from system memory to the NIC.
DPD	Download packet descriptor.
EEPROM	The type of PEROM used in 3C359 NICs to hold configuration information.
field	Two or more adjacent bits; for example, [3:0] is a field of bits from bit 3 down to bit 0.

Term or Acronym	Meaning
flash ROM	The type of PEROM used in 3C359 NICs to hold BIOS code.
FSH	Frame start header.
indication	The reporting of any interesting event on the NIC. Any indication may be configured to cause an interrupt.
interrupt	The actual assertion of the host machine's interrupt signal.
LLC	Logical link control. LLC packets typically originate and terminate in the host. In this document, they are also called host packets.
MAC	Media access control. MAC packets are used to implement the token ring access protocol and are usually local to the token ring subsystem. The driver and host software do not need to process them.
microcode	Code that runs on the NIC communications processor (CP). This code and the associated ASIC implement the token ring protocol. Microcode is firmware held in ROM.
MMIO	Memory-mapped I/O. Memory locations embedded in silicon are used to perform I/O control functions. MMIO is preferred over I/O ports because memory operations on the PCI bus usually execute faster than do I/O operations.
NIC	Network interface card.
NOS	Network operating system.
PEROM	Programmable and erasable read-only memory.
PIO	Programmed I/O.
PHY	IEEE designation for physical layer.
private memory	The memory the NIC uses for data and firmware. It is implemented in the 3C359 NIC with two 32 KB static RAMs (SRAMs) for a total of 64 KB. Private memory is also known as local memory.
Remote Wake-Up	The ability to turn on networked computers from a central location. The computers must have power management capabilities.
RPL	Remote program load. The BIOS code in the optional flash ROM performs RPL, which allows the computer to be booted from the network rather than from its own disk.
SRAM	Static RAM.
SRB	System request block, a control block in shared RAM that passes transmit information or commands from the host to the CP.
UPD	Upload packet descriptor.
upload	The process of transferring received data from the NIC to system memory.
word	A 16-bit wide quantity of data (2 bytes).

(2 of 2)

Register Descriptions and Bit Maps

The register descriptions in this technical reference include register bit maps. For example:



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

The first row of a bit map shows the bit numbers.

The second row of the bit map indicates the following information:

- Shaded areas indicate one of the following:
  - Read-only bits. These bits read back the default values shown. If no value is shown, the read-back value varies.
  - Unimplemented, reserved bits. These bits may be placeholders for possible use in a future revision of the NIC, or they may provide diagnostic information. Reserved bits are writable, but they do not control any function. They disregard data written to them and return zeros when they are read. To maintain compatibility with future versions of the NIC, drivers should write zeroes to reserved bits.
- Unshaded areas indicate active bits. The functions of these bits are described in the register descriptions. A value in an unshaded bit indicates that the driver must write that value to the bit.
- Vertical lines mark the boundaries of fields of bits (for example, [12:0]).

Default bit values are indicated as follows:

- 0 and 1 are known default states.
- x is a bit that is not initialized at reset; thus, its value varies.

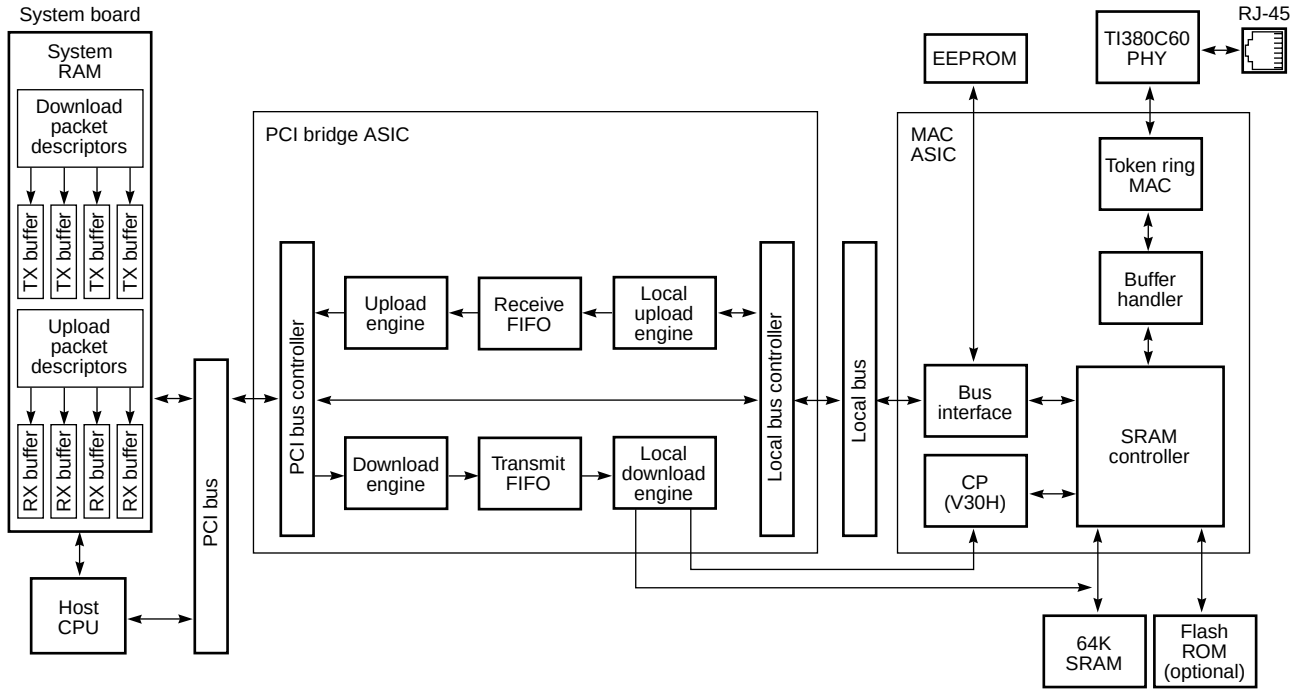
2 ARCHITECTURE

This chapter describes the 3C359 NIC architecture and shows the layout of the registers that the driver can control.

Block Diagram

The block diagram for the 3C359 NIC is shown in Figure 1. The NIC devices are described in the following sections.

Figure 1 3C359 NIC System Architecture



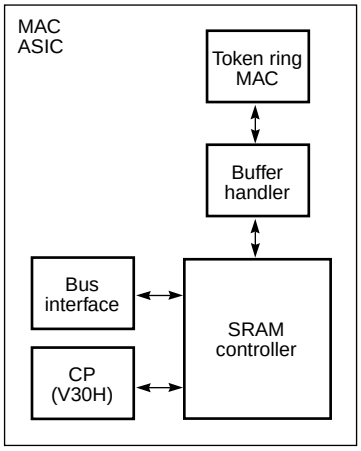
ASICs

The NIC contains two ASICs:

- Media Access Control (MAC) ASIC (see Table 2)
- PCI bridge ASIC (see Table 3)

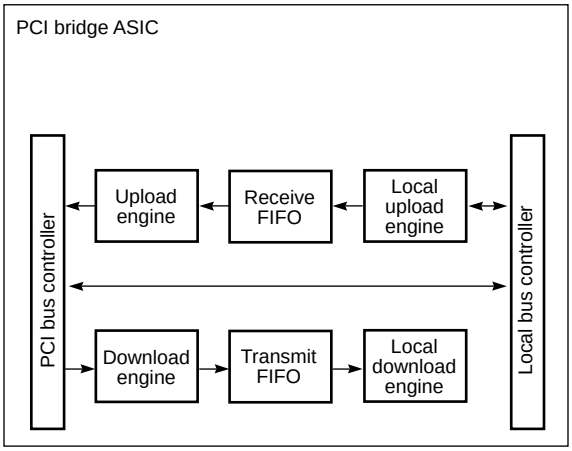
The PCI bridge ASIC connects the PCI bus and the local bus and uses the communications processor (CP) bus connection to write registers in the MAC ASIC directly, thereby controlling MAC ASIC transmissions. Performing this function in hardware rather than with microcode is largely responsible for the 3C359 NIC's very high throughput.

Table 2 MAC ASIC



Function Block	Description
Token ring MAC	This block implements the IEEE 802.5 Media Access Control (MAC) function.
Buffer handler	This block organizes frames within the SRAM.
Bus interface	This block provides the interface between the MAC ASIC and the local bus.
CP (V30H)	This block is the V30H communications processor.
SRAM controller	This block arbitrates among various requesters for memory access (CP, bus interface, and buffer handler) and implements the interface between the MAC ASIC and the SRAM and flash ROM.

Table 3 PCI Bridge ASIC



Function Block	Description
PCI bus controller	This block implements the PCI interface functions (responding to PCI target cycles, generating and controlling PCI master cycles, and performing parity checking and generation). The PCI bus controller also provides bus master services to the download and upload engines.
Upload and download engines	These blocks fetch the descriptors in the downlist and uplist and perform bus master data transfers by requesting PCI bus master burst service from the PCI bus controller block.
	The upload engine removes receive data from the receive FIFO and supplies it to the PCI bus controller as it is required. The download engine pipes transmit data from the PCI bus controller into the transmit FIFO.
Receive and transmit FIFOs	These blocks are high-speed burst caches.
Local upload and download engines	These blocks move data to and from the local bus with service from the local bus controller block.
Local bus controller	This block controls data transfers to and from the local bus.

Other NIC Devices

The other devices associated with NIC operation are described in the following sections.

Flash ROM

The 3C359 NIC has a socket for an optional 128 KB flash ROM that stores microcode and BIOS code. The optional flash ROM is described in "Flash ROM" in Chapter 3.

EEPROM

The 16-bit × 64-word serial EEPROM stores configuration information for the NIC. The EEPROM contents are described in Chapter 5.

64K SRAM

The 64K static RAM (SRAM) provides buffer storage for receive and transmit frames and for firmware. The host software downloads firmware each time the system resets.

Host Registers

The 3C359 NIC presents a set of registers to the host CPU. These host registers are mapped into 128 bytes of the host's I/O space, memory space, or both. A register that is mapped into memory space is called a memory-mapped I/O (MMIO) register.

Because PCI memory transactions typically execute much faster than I/O operations, it is usually best to access the registers in memory space. However, the register set is also mapped into I/O space for the following reasons:

- When memory resources are scarce, the only way to operate the NIC may be through I/O.
- If you need absolutely synchronous control of the NIC, then you must operate through I/O.

In general, host registers must be accessed as operands that are no wider than the bit width of the register. Specific register access limitations are described in the register definitions in this technical reference.

A host register's location is specified by its offset from a base address that is defined in the BaseAddress1 and BaseAddress2 PCI configuration registers. These registers are described in "PCI Configuration Registers" in Chapter 4.

Command Register

Many of a driver's interactions with the NIC are performed using a command structure. Command codes written to the NIC perform some action. For example, the DnEnable command causes the download engine to download frames to the MAC ASIC.

Commands are written to the write-only Command register, which appears at offset 5E. For details on the commands, see Chapter 9.

Commands and status are also exchanged between the host driver and the NIC firmware by means of the software interface. The software commands are described in Chapter 11.

Interrupt Status Register

The read-only IntStatus register shares the location offset 5E with the write-only Command register. A driver uses IntStatus to determine the sources of interrupts on the NIC. Some commands, such as DnStall, initiate a process that may take some time to finish. The IntStatus register includes a bit that indicates when a command issued to the Command register is in the process of being executed. For details on the interrupt status registers, see Chapter 8.

Register Layout Table 4 shows the 3C359 host register layout. Shaded areas in the table indicate reserved spaces. Do not program in these spaces.

Table 4 3C359 Host Register Layout

Byte 3	Byte 2	Byte 1	Byte 0	Offset (Hex)
IntStatus/Command		IndicationEnable		5C
InterruptEnable		TxStartThresh		58
IntStatusAuto				54
				50
				4C
				48
				44
		DnBurstThresh	UpBurstThresh	40
		UpPoll		3C
UpListPtr				38
Countdown		FreeTimer		34
UpPktStatus				30
		DnPoll	DnPriReqThresh	2C
		Config	HashFilter	28
DnListPtr				24
DmaCtrl				20
				1C
	Timer			18
MacAccessCmd				14
		MacData		10
				0C
				08
				04
				00

MAC ASIC Registers

MAC ASIC registers are not mapped directly into host I/O space. They must be accessed using the MacAccessCmd and MacData host registers, as described in Chapter 3.

The main method for host software to communicate with the NIC is through a group of private memory locations, called *control blocks*. See "Communication with the Host" in Chapter 11 for the means by which the locations of these blocks are communicated to the host. You can access control blocks through the MacAccessCmd and MacData host registers.

Table 5 shows the 3C359 MAC ASIC register layout. The memory-mapped I/O (MMIO) registers and the CpAttention I/O register are accessed with the MmioRead and MmioWrite commands in the MacAccessCmd register. The I/O registers, except CpAttention, are accessed with the PrivateMemRead and PrivateMemWrite commands in the MacAccessCmd register.

Shaded areas in the table indicate reserved spaces. Do not program in these spaces.

Table 5 3C359 MAC ASIC Register Layout

	Byte 1	Byte 0	Local Address (Hex)
MMIO Registers	WRBR		CDE02
	WWOR		CDE04
	WWCR		CDE06
	MacStatus		CDE08
	MISR		CDE0B
	RxBufArea		CDE10
	RxEarlyThresh		CDE12
I/O Registers	CPAttention		180D
	Pmbar		1C80
	SwitchSettings		1C88
	EeControl		1C8A
	EeData		1C8C

3 OPERATION

This chapter summarizes NIC operational characteristics.

Software Interface

The software interface allows a driver to perform high-level operations such as inserting the NIC into the ring (opening the NIC) or requesting statistics from the MAC ASIC. The software interface is described in Chapter 11.

Statistics

The NIC accumulates statistics to support network management software. The host retrieves these statistics with the Get.Statistics command, described in Chapter 11.

Flash ROM

The 3C359 NIC has a socket to hold an optional flash ROM (also called a boot ROM). The flash ROM contains BIOS code that allows the host computer to be booted remotely over the network, rather than from its own disk. It also contains a copy of the NIC microcode, which is normally held in SRAM. Placing microcode in the flash ROM frees SRAM, allowing additional data buffer space, which is an important consideration in server applications.

The flash ROM is a 64K × 16-bit flash device that can be written electrically with a flash write utility. The flash ROM and utility are available together from 3Com (part number 3C359-TRIROM).

The flash ROM size is 128 KB. The top 64 KB is reserved for microcode; the bottom 64 KB is reserved for BIOS code. The flash ROM is configured through the BiosRomControl PCI configuration register. This register causes the ROM to be mapped into the memory space of the host system, allowing the ROM contents to be scanned, copied to system RAM, and executed at initialization time.

The 3C359 NIC uses the enBios bit in the Pmbar register to decide how to respond to accesses to the flash ROM space defined in the BiosRomControl register. Also, if either the addressDecodeEnable bit in the BiosRomControl register or the memorySpace bit in the PciCommand register is cleared, then the PCI bridge ASIC does not respond to accesses to flash ROM space.

Table 6 shows the possible bit combinations for flash ROM access (cases A, B, C, and D) and the 3C359 NIC's responses in each case.

Table 6 Responses to Flash ROM Access

Case	Bits: memory Space*	Bits: address Decode Enable†	Bits: enBios‡	NIC Response
A	0	x	x	No response to memory access. The NIC does not activate the DEVSEL# signal on the PCI bus.
B	1	0	x	Responds only to memory accesses to the area defined by the BaseAddress2 register (MMIO).
C	1	1	0	Responds to memory accesses as in case B, but returns ones for memory accesses to the area defined by the BiosRomControl register.
D	1	1	1	Responds to MMIO accesses as in case B and returns ROM data for a memory access to the area defined by BiosRomControl. The MAC ASIC's Pmbar register must be set to F9C0h to read the ROM correctly. The EEPROM must be written with this private memory base address when the ROM is installed.

* The memorySpace bit is located in the PciCommand register.
† The addressDecodeEnable bit is located in the BiosRomControl register.
‡ The enBios bit is located in the MAC ASIC Pmbar register.

Data Structure Lists

To move data between the host and the NIC, drivers set up data structures in system RAM to specify the buffers to be used for packet data movement. These data structures, called descriptors, are linked together in system memory to form lists.

All packet data is moved across the 3C359 PCI bus by bus master operations. The 3C359 NIC also uses bus master operations to read descriptor information out of system RAM and to write status back into the descriptors.

Movement of a transmit packet to the NIC is called a download. The list of download packet descriptors (DPDs) is called the *downlist*. Similarly, a receive packet movement is called an upload, and the list of upload packet descriptors (UPDs) is the *uplist*.

The driver creates and maintains the uplist and downlist. It starts the download process by writing the address of the first download descriptor in the downlist to the DnListPtr register. Uploads are started by writing the first upload descriptor address to the UpListPtr register. The driver also accesses NIC registers for initialization, interrupt handling, statistics collection, and error handling. For details on data structure lists, see Chapter 6 and Chapter 7.

PCI Bus Master Operation

This section describes aspects of bus master operation that can be controlled by software. For information about PCI configuration, see "PCI Configuration Registers" in Chapter 4.

An on-board bus mastering mechanism passes data to and from the host. Independent, full-fragment gathering (download) and scattering (upload) DMA engines allow full-duplex operation and reduce the amount of buffer RAM required on the NIC. The DMA engines reference DPDs and UPDs in host memory. UPDs and DPDs indicate the size and location of the buffers for each packet. When the host transmits a packet, the driver programs the location of the DPD into the NIC to trigger the DMA engine to begin a download.

Likewise, packets move from the NIC to host memory according to a UPD. A UPD can be specified before the NIC receives the packet (in which case, the packet moves immediately off the wire and into host buffers), or can be specified after a packet has been received. The former case results in higher performance.

The UPD can specify buffer sizes ranging from just large enough to hold some amount of look-ahead data to large enough to hold a maximum size packet. The software environment determines the most appropriate arrangement to use.

PCI Memory Commands

The 3C359 NIC supports all PCI memory commands. (See Table 7.)

Table 7 PCI Memory Commands

Command	Description
MW	Memory write
MWI	Memory write invalidate
MR	Memory read
MRL	Memory read line
MRM	Memory read multiple

To read packet data transmitted to it, the NIC uses the MR, MRL, or MRM command. To write packet data it receives, the NIC uses either the MW or MWI command. For maximum bus efficiency, the NIC decides which command to use on a burst-by-burst basis. The choice depends on the remaining number of bytes in the fragment, the amount of free space in the transmit or receive FIFO, and certain system parameters, such as cache line size.

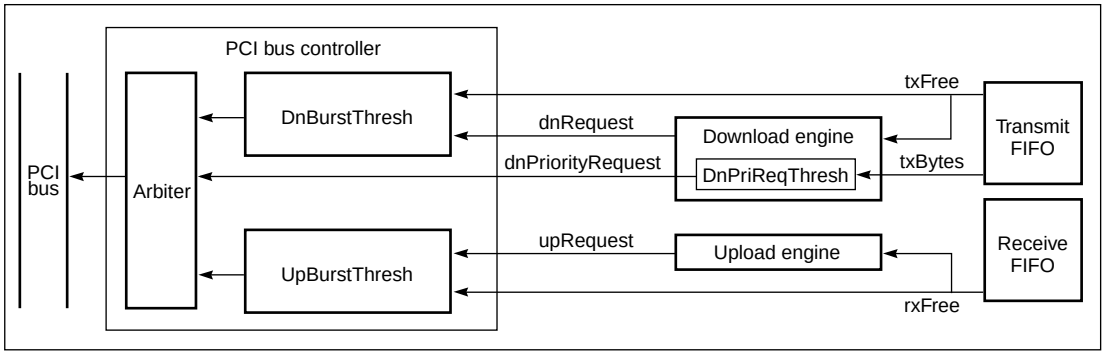
These configuration bits control the use of PCI memory commands:

- The MWIEnable bit in the PciCommand configuration register enables or disables the NIC's use of MWI.
- The defeatMWI bit in the DmaCtrl register can disable the NIC's use of MWI, independently of the MWIEnable bit. By default, MWI is enabled.

PCI Bus Request Control

A set of registers controls PCI burst behavior. These registers allow trade-offs to be made between PCI bus efficiency and underrun and overrun frequency. Figure 2 illustrates the bus request structure.

Figure 2 3C359 NIC Bus Request Structure



Arbitration logic (the arbiter) within the PCI bus controller block accepts bus requests from the download and upload engines.

Download

The download engine monitors the amount of free space in the transmit FIFO. When there are at least 16 bytes of free space and a fragment available for download, the download engine issues the dnRequest signal to make a standard bus request. The DnBurstThresh logic qualifies dnRequest. When the amount of free space in the FIFO is greater than the value in the DnBurstThresh register, a download request is passed on to the arbiter. The purpose of DnBurstThresh is to delay the bus request until there is enough free space in the FIFO for a long, efficient burst.

The download engine also has a way to make an emergency bus request. When the number of used bytes in the FIFO drops below the value in the DnPriReqThresh register, indicating that the FIFO is approaching an underrun condition, the dnPriorityRequest signal makes a priority request. This request is not subject to the DnBurstThresh constraint; when the FIFO is close to underrun, burst efficiency is sacrificed in favor of requesting the bus as quickly as possible.

Upload

The upload mechanism is similar to download. The upload engine monitors the number of bytes in the receive FIFO. When there are at least 16 bytes in the FIFO and a buffer is available for upload, the upload engine issues the upRequest signal to make a standard bus request. The UpBurstThresh logic qualifies the upRequest signal; when the number of bytes in the FIFO is greater than the value in the UpBurstThresh register, an upload request is passed on to the arbiter.

Upload does not have a priority request mechanism because the upload FIFO buffer is larger than 30 KB, which is large enough to tolerate large bus latencies. Download, with only 512 bytes of buffering, requires the priority mechanism.

Power Management

The NIC supports power management directed by the operating system, in accordance with the *Advanced Configuration and Power Interface (ACPI)* specification. A properly equipped PC can put itself into a low-power state while the NIC remains active on the ring. A defined network event can then be used to wake the computer remotely. This mode is known as *remote wake-up* mode.

3C359 NICs include power management registers in the PCI configuration space, as defined by the *PCI Bus Power Management Interface Specification, Revision 1.0*. The PowerMgmtCap register supplies the system with information about the NIC's power management support and capabilities. The PowerMgmtCtrl register allows system or driver software to read the NIC's power management status and set the NIC's power state.

For a complete discussion of power management, refer to the *PCI Bus Power Management Interface* specification on the World Wide Web at www.pcisig.com.

Remote Wake-up Mode

When the NIC prepares to enter the remote wake-up mode, the driver specifies one or more types of wake-up packets. While in this mode, the NIC does not process DPDs, nor does it upload to the host any logical link control (LLC) frames

that it receives. It simply monitors the ring for a wake-up packet. Firmware parses all received frames, looking for one that matches the contents of one of the wake-up packet types. Packets that do not match are discarded.

When the NIC recognizes a remote wake-up packet, it activates a signal on the PCI bus connector that causes the PC to start up and assume an operational state. Firmware takes the following actions when it receives a remote wake-up packet:

1. Puts an appropriate response into the adapter request block (ARB), indicating that it has received a remote wake-up packet.
2. Saves the remote wake-up packet for possible forwarding to the host. The driver needs to reconfigure the NIC for normal operation before the packet can be forwarded.

Power States

Table 8 defines the supported power states. The current power state is determined by the powerState field in the PowerMgmtCtrl register.

Table 8 Power States

State	powerState Value	Description
$D0_{uninitialized}$	0	$D0_{uninitialized}$ is the result of a hardware reset, or of a transition from $D3_{hot}$ to D0. The PCI configuration registers are uninitialized, and the NIC responds to PCI configuration cycles only.
$D0_{active}$	0	$D0_{active}$ is the normal operational power state for the NIC. In $D0_{active}$, the PCI configuration registers have been initialized by the system, including the ioSpace, memorySpace, and busMaster bits in the PciCommand register. Therefore, the NIC is able to respond to PCI I/O and memory and configuration cycles and can operate as a PCI master. The NIC cannot signal wake-up (assert the PME# signal on the PCI bus) from the D0 state.
D1	1	D1 is a "light-sleep" state, which allows transition back to D0 with no delay. In D1, the PCI clock is running. The NIC responds to PCI configuration accesses, allowing the system to change the power state, but it does not respond to PCI I/O or memory accesses. The NIC's function in the D1 state is to recognize wake-up events and link state events and pass them on to the system by asserting the PME# signal on the PCI bus.
D2	2	D2 is a partial power-down state that allows a faster transition back to D0 than is possible from the D3 state. D2 is functionally identical to D1, except that in D2, the PCI clock may be stopped, reducing power consumption even further.
$D3_{hot}$	3	$D3_{hot}$ is the full power-down state for the NIC. In $D3_{hot}$, the NIC shuts down and places itself into the lowest power condition. In $D3_{hot}$, the NIC responds to PCI configuration accesses, to allow the system to change the power state back to $D0_{uninitialized}$, but it does not respond to PCI I/O or memory accesses.
$D3_{cold}$	N/A	This is the power-off state for the NIC. The NIC has no function in this state. Restoring power causes a hardware reset, which puts the NIC into the $D0_{uninitialized}$ state.

The system puts the NIC into either the D1 or D2 state when it is going to power down but wants the NIC to monitor for wake-up packets. From the NIC's perspective, D1 and D2 are identical, but D2 consumes less power because the PCI clock may be stopped.

Because the clock is stopped in D2, the dynamic power consumption of the PCI bridge ASIC drops dramatically, thus reducing the overall power consumption of the NIC. Only as much logic as is necessary to allow the NIC to monitor for wake-up packets and to assert the PME# signal is left powered. Because the NIC is still inserted in the ring and participating in ring polling, power consumption of the rest of the NIC is unaffected by whether it is in the D0, D1, or D2 state.

Exiting the D1 or D2 states involves changing the powerState bit in the PowerMgmtCtrl register.

In $D3_{hot}$ the PCI bridge ASIC turns the 32-MHz clock off. The RESETOUT signal is applied to the MAC control ASIC and the PWRDN signal is applied to the TI380C60 ring interface device (PHY). Power consumption in this state is less than a few tens of milliwatts, most of which is consumed in the 5-V–tolerant I/O buffers between the PCI bridge ASIC and the MAC control ASIC, and in the 3.3-V regulator. In the $D3_{hot}$ state, it is not possible to monitor for wake-up packets.

Removing power puts the NIC into the $D3_{cold}$ state, and all NIC context is lost. $D3_{cold}$ can only be exited by restoring the power and clock and then asserting the RST# signal.

In accordance with the PCI specification, the NIC ignores all bus transactions except configuration accesses when it is in the D1, D2, or $D3_{hot}$ states.

Programming Remote Wake-up Events

The following steps put the 3C395 NIC into the remote wake-up mode (D1 or D2):

1. The operating system calls the driver and tells it to go into remote wake-up mode.
2. The driver takes the following actions:
   - **a** Waits until all queued download packets have been transmitted, or stalls the download and purges all remaining packets from memory.
   - **b** Uses the Set.Sleep.Mode command to tell the firmware that it is to enter the remote wake-up mode. The firmware reconfigures the MAC receiver to place all LLC frames that pass the address filter into private MAC buffers. The driver specifies to the firmware the contents of all remote wake-up packet types to be monitored.
   - **c** Processes or discards all receive frames still in the queue. The host also processes all pending interrupts.
   - **d** Informs the operating system that the NIC is ready to be put to sleep.
3. The operating system puts the NIC into either the D1 or D2 state and uses the PowerMgmtCtrl register to enable the PME# pin.

The following steps put the 3C395 NIC into the full power-down state ($D3_{hot}$):

1. The driver closes the NIC (remove it from the ring) and informs the operating system that the NIC is closed.
2. The operating system puts the NIC in the $D3_{hot}$ state.
3. The PCI bridge ASIC initiates a controlled power-down sequence:
   - **a** Sets the PCI bridge ASIC's pins to predetermined power-down states.
   - **b** Applies the RESETOUT signal to the MAC control ASIC.
   - **c** Applies the PWRDN signal to the PHY.
   - **d** Shuts off the 32-MHz master clock.

   This process requires 32 PCI clock cycles. After 32 cycles, the PCI clock can be stopped.

When the PCI clock is restarted, the NIC exits the $D3_{hot}$ state and enters the D0 state through the PowerMgmtCtrl register. When exiting $D3_{hot}$, the PCI bridge ASIC automatically issues a GlobalReset command and performs autoinitialization, returning the NIC to a state equivalent to that after a hardware reset (as if the RST# signal had been asserted).

Accessing and Managing Private Memory

Private memory (also known as local memory) is the memory the NIC uses for microcode, data, and command and status blocks. The minimum amount of private memory is 64 KB; the maximum is 1 MB. The 3C359 NIC implements 64 KB of SRAM in its standard SRAM-based configuration and 192 KB (64 KB SRAM, 128 KB ROM) when the optional flash ROM is installed.

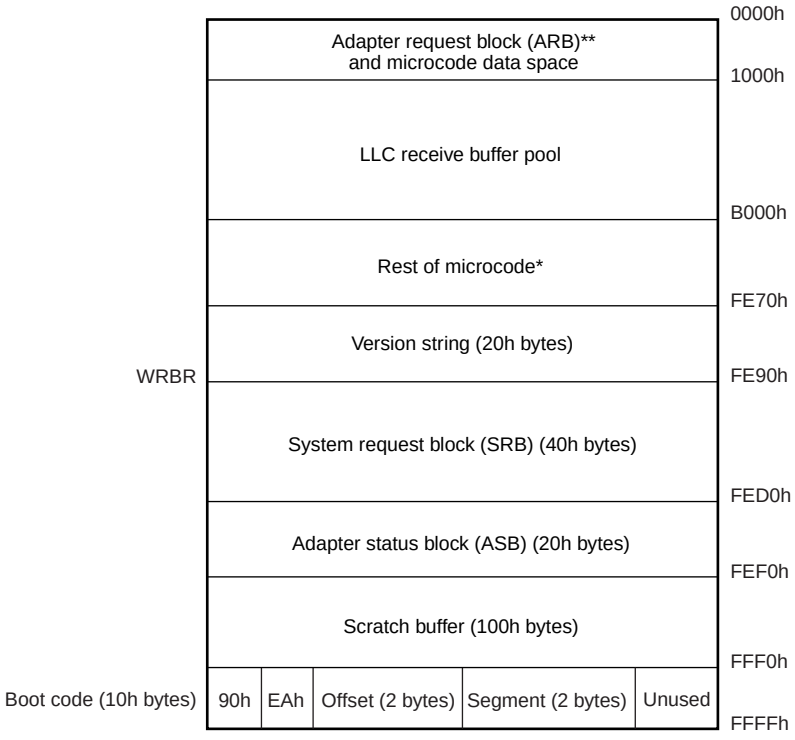
Memory Usage

Memory usage depends on whether or not flash ROM is installed.

Without Flash ROM

When no flash ROM is installed, the microcode resides in the RAM and allocates approximately 139 LLC receive buffers. The receive buffer size is fixed at 256 bytes. Memory usage in this RAM-based configuration is shown in Figure 3.

Figure 3 RAM-Based Configuration Memory Usage



* Assume mcrocode size is 5000h bytes

**ARB is located at 08A0h

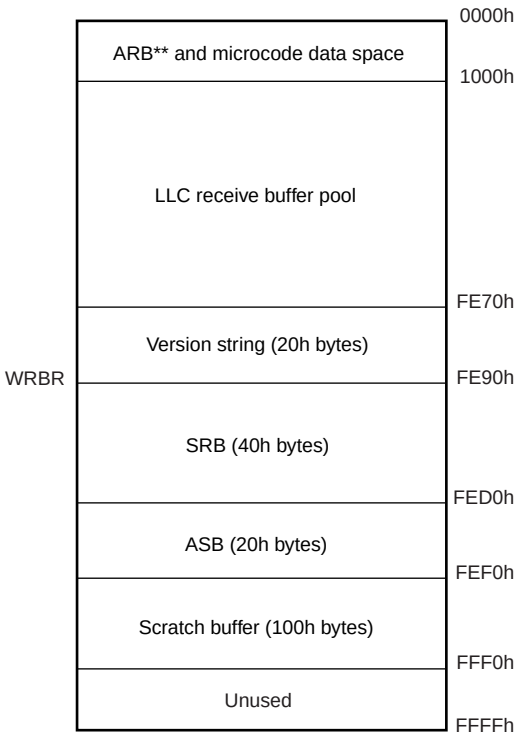
As shown in Figure 3, one read/write region is defined (FE90h to FFFFh). The rest of the RAM is write-protected. The microcode preserves itself after the initialization process; therefore, the driver need not download microcode each time it holds the CP (using the cpHold bit in the Pmbar register).

With Flash ROM

When flash ROM is installed, the microcode can allocate approximately 238 LLC receive buffers. The receive buffer size is fixed at 256 bytes.

Memory usage with flash ROM installed is shown in Figure 4. One read/write region is defined (FE90h to FFFFh). The rest of the RAM is write-protected.

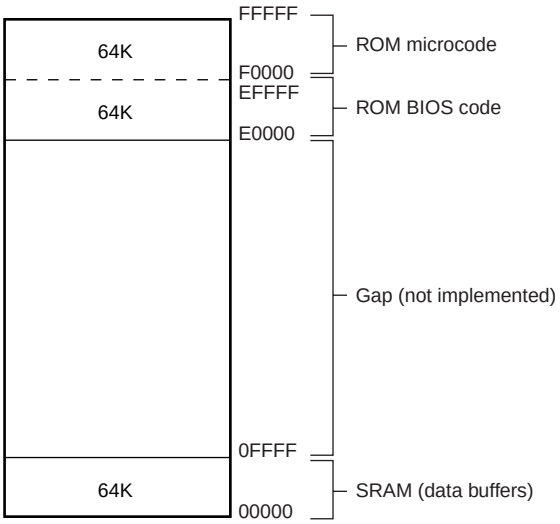
Figure 4 Memory Usage with Flash ROM Installed



**ARB is located at 08A0h

Memory Access As shown in Figure 5, the SRAM is located at private memory addresses 00000h through 0FFFFh. The ROM is located at addresses E0000h through FFFFFh.

Figure 5 Private Memory Partitioning



Only 64 KB of private memory is accessible to the host through the MacAccessCmd and MacData registers at any given time. The privateMemoryBase field in the Pmbar register is used to select which 64K block of private memory is to be accessible. For example, to access SRAM, privateMemoryBase is set to 0'b0000000; to access the first 64 KB of ROM, it is set to 0'b1110000; to access the second 64 KB of ROM, it is set to 0'b1111000. Normally, privateMemoryBase is set to 0 so that the SRAM partition is accessible. Other portions of private memory are selected temporarily to program the ROM or to perform remote boot.

The SRAM is based at DD000h in private memory space. This base address must be added to the offset of the location you want to access. The result is used in the localAddress field of the MacAccessCmd register. For example, to read the byte of SRAM offset 101h, add D0000h and 101h (equals D0101h) and place this value into the localAddress field of MacAccessCmd along with the opcode for a private memory read (0'b10000). Then read the result from the MacData register. Remember that privateMemoryBase must be set properly (to 0'b000000000 in this case) before MacAccessCmd is written.

The read/write and read-only areas of private memory have different boundaries. The private memory management registers—WRBR, WWCR, and WWOR—define the read/write and read-only boundaries.

The write region extends from the highest address of the private memory to a variable origin specified by the WRBR register.

The write window extends from a variable base defined by the WWOR register pair to a variable limit defined by the WWCR register pair.

The least-significant bit in each odd register is 0, because all write boundaries are word-aligned.

Until the NIC has been opened, the host only has read-only access to private memory. After it is opened, the NIC indicates which regions are writable through the WRBR, WWCR, and WWOR memory management registers.

The registers associated with accessing and managing private memory are described in the following sections.

CPAttention

Synopsis	Provides resources for firmware development.
Type	Read/write
Size	8 bits
Local Address	180Dh

CPAttention Register Format

7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0

CPAttention Bit Descriptions

Bit	Name	Description
[7]	pmbarVisible	Makes the Pmbar register writable after a reset. Normally, Pmbar is read-only after a reset.
[6]	memWrEn	Makes all of shared memory writable by the host. This overrides any settings in the WWBR, WWOR, and WWCR registers and allows CP code to be downloaded to RAM (on RAM-only NICs) regardless of the write-protection provided by these registers. This bit can also be used by host-based diagnostics for testing RAM.

MacAccessCmd

Synopsis	Used in conjunction with the MacData register to access MAC ASIC registers and private memory.
Type	Read/write
Size	32 bits
Offset	14

MacAccessCmd Register Format

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

The MacAccessCmd register works in conjunction with the MacData register to access private memory and MAC ASIC registers.

To read or write information, first specify the command and address with the MacAccessCmd register and then read the result from, or write the data to, the MacData register.

***CAUTION**: The command specified in MacAccessCmd and the operation performed on MacData must be consistent—if a read is specified with MacAccessCmd, MacData must be read; if a write is specified, MacData must be written. Mixing different types of MacAccessCmd commands and MacData operations causes a PCI target abort and is likely to corrupt data in the MAC ASIC registers.*

The parameter in bit 27 specifies the following:

- 0 = byte access
- 1 = word access

The following sections describe MacAccessCmd commands. The command definitions use the following conventions:

- The bit value is the 32-bit value that the NIC expects to be written to the MacAccessCmd register to carry out the desired operation.
- The opCode field (bits [31:26]) specifies whether an I/O, MMIO, or SRAM read or write access is to be done, and whether it is a byte or a word access.
- The localAddress field (bits [19:0]) specifies the address that the PCI bridge ASIC is to apply to the MAC ASIC address bus during the access.
- Bit positions occupied by an "X" indicate that the value for the corresponding bit does not matter. However, for future hardware compatibility it is recommended that zeros be written to these positions.
- Bit positions occupied by a dot (•) indicate bit positions that are to be filled by the parameter associated with the command.

PrivateMemRead

Bit Value	(1000 •1XX XXXX •••• •••• •••• •••• ••••)

Used to read locations in SRAM or ROM. The localAddress field (bits [19:0]) is the 20-bit address of the SRAM or ROM location to be read. Private memory locations to be accessed must be visible within SRAM. If a location outside SRAM is to be accessed, the privateMemoryBase bit in the Pmbar register must first be set accordingly. To read the addressed location, simply read the MacData register.

Accesses can be either 8- or 16-bit. Accesses of 24 and 32 bits are not permitted.

PrivateMemWrite

Bit Value	(0100 •1XX XXXX •••• •••• •••• •••• ••••)

Used to write locations in SRAM or ROM. The localAddress field (bits [19:0]) is the 20-bit address of the SRAM or ROM location to be written. Private memory locations to be accessed must be visible within SRAM. If a location outside SRAM is to be accessed, the privateMemoryBase bit in the Pmbar register must first be set accordingly. The PrivateMemWrite command should be written first, followed by loading the write data into the MacData register. An access to MacData triggers the write operation to the addressed location.

MmioRead

Bit Value	(1000 •0XX XXXX •••• •••• •••• •••• ••••)

Used to read MMIO registers. The localAddress field (bits [19:0]) is the 20-bit address of the MMIO location to be read. To read the addressed location, simply read MacData.

Accesses can be either 8- or 16-bit. Accesses of 24 and 32 bits are not permitted.

MmioWrite

Bit Value	(0100 •0XX XXXX •••• •••• •••• •••• ••••)

Used to write MMIO registers. The localAddress field (bits [19:0]) is the 20-bit address of the MMIO location to be written. The MmioWrite command should be written first, followed by loading the write data into the MacData register. An access to MacData triggers the write operation to the addressed location.

Accesses can be either 8- or 16-bit. Accesses of 24 and 32 bits are not permitted.

Some MMIO registers are 8-bit and others are 16-bit. When the driver writes MMIO registers, the command used and the register width must be consistent. Table 9 summarizes the MMIO register sizes and locations.

Table 9 MMIO Register Sizes and Locations

Register	Size	Local Address (Hex)
WRBR	Word	cde02
WWOR	Word	cde04
WWCR	Word	cde06
MISR	Byte	cde0b
RxBufArea	Word	cde10
RxEarlyThresh	Word	cde12

IoRead

Bit Value	(0010 XXXX XXXX XXXX •••• •••• •••• ••••)

Used to read I/O registers in the MAC ASIC. The localAddress field (bits [19:0]) is the 16-bit address of the I/O location to be read. Register read data is returned in the MacData register.

Accesses can be either 8- or 16-bit. Accesses of 24 and 32 bits are not permitted.

IoWrite

Bit Value	(0001 •XXX XXXX XXXX •••• •••• •••• ••••)

Used to write I/O registers in the MAC ASIC. The localAddress field (bits [19:0]) is the 16-bit address of the I/O location to be written. The IoWrite command should be written first, followed by loading the write data into the MacData register.

Accesses can be either 8- or 16-bit. Accesses of 24 and 32 bits are not permitted.

Some I/O registers are 8-bit and others are 16-bit. When the driver writes I/O registers, the command used and the register width must be consistent. Table 10 summarizes the I/O register sizes and locations.

Table 10 I/O Register Sizes and Locations

Register	Size	Local Address (Hex)
CPAttention	Byte	180d
Pmbar	Word	1c80
EeControl	Word	1c8a
EeData	Word	1c8c

MacData

Synopsis	Works in conjunction with the MacAccessCmd register to provide read-write access to MAC ASIC registers and memory through PCI bridge ASIC register space.
Type	Read/write
Size	16 bits
Offset	10

MacData Register Format

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

MacData contains the value accessed with the MacAccessCmd register. MacAccessCmd specifies whether memory or I/O is to be read or written. The command is specified first using MacAccessCmd, followed by a corresponding read from or write to MacData.

***CAUTION**: The command specified in MacAccessCmd and the operation performed on MacData must be consistent—if a read is specified with MacAccessCmd, MacData must be read; if a write is specified, MacData must be written. Mixing different types of MacAccessCmd commands and MacData operations causes a PCI target abort and is likely to corrupt data in the MAC ASIC registers.*

Some reads, such as ROM reads, do not finish immediately. The NIC forces PCI retries until the read data is available.

When a byte-read access is made to either an even or an odd offset, read data is always placed in bits [7:0] of the MacData register, and bits [15:8] are undefined. When doing a byte write access to either an even or an odd offset, the data to be written should always be placed in bits [7:0] of MacData, and bits [15:8] are "don't care."

Pmbar

Synopsis	Private memory base address register that determines the region of private memory accessible with the PrivateMemRead and PrivateMemWrite commands.
Type	Read/write
Size	16 bits
Local address	1C80h

Pmbar Register Format

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

Pmbar Bit Descriptions

Bit	Name	Description
[11]	enBios	When a flash ROM is on the NIC and remote program load is desired, autoconfiguration sets this bit to 1, which enables the NIC to map the boot BIOS code into host memory space (according to the PCI configuration).
[10]	cpHold	When no flash ROM is on the NIC, autoconfiguration sets this bit to 1, which keeps the CP in a reset state until the driver releases it (presumably after microcode has been downloaded). When a ROM is on the NIC, this bit is 0, permitting immediate operation of the NIC.
[9]	wdtd	When set, this bit disables the NIC's internal watchdog timer. This bit should only be set in a development environment. The default is cleared, which enables the watchdog timer.
[8:2]	privateMemoryBase	This field is used to page private memory into the private memory window. When the NIC has no flash ROM installed, the privateMemoryBase bit initializes to 00h. When flash ROM is installed and remote boot is desired, privateMemoryBase initializes to 1C0h. Boot code is responsible for setting privateMemoryBase back to 00h in preparation for opening the NIC.

WRBR

Synopsis	Determines the address of the beginning of the private memory write region.
Type	Read-only
Size	16 bits
Local address	CDE02h

WRBR Register Format

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

The host can write to private memory between and including the address specified in the WriteRegionBase register (WRBR) and the top of the window.

Bits [15:8] define the least-significant byte of WRBR. Bits [7:0] define the most-significant byte. Any attempt by the host to write this register causes an access interrupt, unless bit 6 in the CPAttention register is a 1.

Bit 8 is always 0.

WWCR

Synopsis	Defines the end of the private memory write window.
Type	Read-only
Size	16 bits
Local address	CDE06h

WWCR Register Format

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

Bits [15:8] of the WriteWindowClose register (WWCR) define the least-significant byte. Bits [7:0] define the most-significant byte.

The offset specified by WWCR is read-only. The offset just before WWCR is the last writable location in the write window.

Any attempt by the host to write this register causes an access interrupt, unless bit 6 in the CPAttention register is a 1.

Bit 8 is always 0.

WWOR

Synopsis	Determines the start of the private memory write window.
Type	Read-only
Size	16 bits
Local Address	CDE04h

WWOR Register Format

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

The host can write to private memory between and including the address specified in the WriteWindowOpen register (WWOR) and up to the WWCR register. Bits [15:8] define the least-significant byte of WWOR. Bits [7:0] define the most-significant byte.

Any attempt by the host to write this register causes an access interrupt, unless bit 6 in the CPAttention register is a 1.

Bit 8 is always 0.

4 CONFIGURATION

This chapter discusses the configuration mechanism for the NIC and defines the registers associated with configuration. Configuration has two components: NIC configuration and PCI configuration.

System Reset

System reset is the assertion of the hardware reset signal on the PCI bus, which causes a complete reset of the NIC, including forcing flip-flops to known values, losing any NIC configuration that had been set, and loading the default configuration from the EEPROM.

There are two sources of system reset:

- A hardware reset, caused by asserting the RESETN signal after power-up, which brings the NIC into a known state
- A software-controlled reset, using the GlobalReset command in the Command register

  A GlobalReset command bit mask parameter allows selective reset of various parts of the NIC.

For details on GlobalReset and other reset commands, see "Reset Commands" in Chapter 9.

Serial EEPROM

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

Shortly after system reset, the NIC ASICs read certain locations from the EEPROM and place the data into the host-accessible registers shown in Table 11. If the EEPROM has the contents specified in Chapter 5, a reset produces the NIC configuration shown in Table 12.

Table 11 EEPROM Data Locations

			Read by:	
EEPROM Location	**Register**	**Value (Hex)**	**MAC ASIC**	**PCI Bridge ASIC**
03	DeviceId (0x3590)	3590	X	X
09	ResourceRedirector	509F	X	
0A	ConfigurationControl	0000	X	

(1 of 2)

Table 11 EEPROM Data Locations (continued)

EEPROM Location	Register	Value (Hex)	Read by: MAC ASIC	Read by: PCI Bridge ASIC
0B	Pmbar	F600*	X	X
0C	PciParms1	FC18		X
17	SubsystemVendorId	10B7†		X
18	SubsystemId	3590‡		X
19	PciParms2	429E		X

(2 of 2)

* The Pmbar value is for the standard configuration, with no flash ROM installed. When a flash ROM is installed and the boot BIOS is enabled, the Pmbar value is 0x19C0. When a flash ROM is installed and the boot BIOS is not enabled, the Pmbar value is 0x1000.

† As shipped from 3Com. May be altered by a value-added reseller.

‡ As shipped from 3Com. May be altered by a value-added reseller.

Table 12 NIC Configuration After System Reset

Item	Configuration
Pmbar visibility	Visible
Private memory writability	Not writable. To download microcode, the loader must set the memWrEn bit in the CPAttention register to make private memory writable. Following the download, the loader should set the memWrEn bit back to 0. The microcode determines private memory writability through the WWOR, WWCR, and WWBR memory management registers.
CP state	Held in the reset state if flash ROM is not installed; not held if flash ROM is installed.
Private memory base address	00000h.
Private memory page size	64K.
Private memory size	64K.

NIC Configuration

The basic NIC configuration steps—autoinitialization, PCI configuration, and driver configuration—are described in the following sections.

Autoinitialization

Following a system reset, the autoinitialization state machine reads configuration settings from the EEPROM and configures the MAC ASIC and the local bus.

These steps configure the NIC after reset:

1 The MAC ASIC reads the following registers from EEPROM:

- DeviceId (3590h)
- SwitchSettings
- ResourceRedirector
- ConfigurationControl
- Pmbar (private memory base address register)

2 The PCI bridge ASIC monitors the process in step 1. When the process is finished, the PCI bridge ASIC activates the MAC ASIC register set and performs the following steps:

   a Reads the DeviceId register from EEPROM and places its contents in the DeviceId PCI configuration register.

   b In the MAC ASIC, writes the CPAttention register with 80h to make the Pmbar register visible.

   c Reads the Pmbar register (for internal use).

   d Reads the SubsystemVendorId register from EEPROM and places it in the SubsystemVendorId PCI configuration register.

   e Reads the SubsystemId register from EEPROM and places it in the SubsystemId PCI configuration register.

   f Reads the PciParms1 and PciParms2 registers from EEPROM and places them in various PCI configuration registers.

   g Writes bits [7:0] in the MAC ASIC RamRelocation register with D0h to set the private memory base address to D0000h.

PCI Configuration

Following autoinitialization, PCI configuration proceeds. The system BIOS performs the following steps:

1 Establishes the PCI configuration for the NIC, including the allocation of memory and I/O resources.

2 Searches for a flash ROM on the NIC.

3 If a flash ROM is installed and it is enabled (see the Pmbar footnote in Table 11), copies the boot image from the flash ROM to system RAM and executes the code from the RAM.

Driver Configuration

After the system has been booted (either remotely if a flash ROM is installed, or from the local disk), the driver is loaded. The driver performs different steps, depending on whether a flash ROM is installed.

Without Flash ROM Installed

Without flash ROM installed, the driver must download the microcode into private memory before the NIC can be opened. Once this is done, the CP (which has been held in the reset state since the system reset) can be released by clearing the cpHold bit in the Pmbar register.

The download process is:

1 In the Pmbar register, set the privateMemoryBase bit to 0.

2 In the CPAttention register, set the memWrEn bit to 1.

This allows the driver to use the MacAccessCmd and MacData registers to write the first 64 KB of private memory. Because the 3C359 NIC contains only 64 KB in its standard configuration, this is sufficient to access all of private memory.

3 Write the PrivateMemWrite command with the appropriate localAddress bit into the MacAccessCmd register. The write operation can be in either byte or word length. If it is word length, the local address you specify must be an even boundary.

4 Write the data into the MacData register.

The PCI bridge ASIC arbitrates for access to the MAC ASIC. When access is granted, the PCI bridge ASIC writes the data to the addressed location.

5 Repeat steps 3 and 4 for each subsequent word to be written.

6 Following the download:

   a In the CPAttention register, clear the memWrEn bit (to protect private memory from inadvertent writes by the host).

   b In the Pmbar register, clear the cpHold bit (to start the processor).

After the driver releases the microcode, the microcode performs a self-test and prepares itself to receive and process commands from the driver. The driver performs these additional configuration steps:

7 Specifies the ring speed, if different from the EEPROM setting. (See "Detecting Ring Speed" in Chapter 11.)

8 Writes the RxBufArea register with D0000h.

9 Specifies the following register values:

- RxEarlyThresh = See "RxEarlyThresh" in Chapter 7.
- TxStartThresh = See "TxStartThresh" in Chapter 6.
- DnPriReqThresh = See "DnPriReqThresh" in Chapter 6.

At this point the NIC may be opened. A complete listing of the EEPROM contents is given in Chapter 5.

With Flash ROM Installed

When the flash ROM is installed, the driver only needs to perform steps 7 through 9 of the "Without Flash ROM Installed" driver configuration procedure described in the previous section to download the microcode.

The optional 3Com flash ROM that can be installed on the 3C359 NIC has these uses:

- It holds the boot BIOS code, thus permitting remote program load (RPL).
- It contains the microcode for the NIC. This frees SRAM space (which would otherwise be used to hold the microcode) and greatly increases the amount of memory available for receive data buffers. For this reason alone, it may be desirable to install the flash ROM, even if RPL is not needed.

  RPL is enabled and disabled by setting the Pmbar field in EEPROM to a value that causes the enBios bit in the Pmbar register to be on or off. For the Pmbar field values, see the Pmbar footnote in Table 15 in Chapter 5. For instructions on changing this value, see the documentation that comes with the flash ROM.

PCI Configuration Registers

PCI NICs use a slot-specific block of configuration registers to perform NIC configuration. PCI configuration cycles are directed at one of eight possible PCI logical functions within a single physical PCI device. The configuration registers are accessed with two types of PCI configuration cycles:

- Type 0 cycles are used to configure devices on the local PCI bus.
- Type 1 cycles are used to pass a configuration request to a PCI bus at a different hierarchical level.

3C359 PCI NICs respond only to Type 0 configuration cycles directed at function 0. The NIC ignores Type 1 cycles and Type 0 cycles that are directed at functions other than 0.

Each PCI device decodes 256 bytes of configuration registers. Of these, the first 64 bytes are predefined by the PCI specification. The remaining registers may be used as needed for PCI device-specific configuration. The 3C359 NIC implements a group of power management registers in this device-specific register space.

In PCI configuration cycles, the host system provides a slot-specific decode signal (IDSEL), which informs the NIC that a configuration cycle is in progress. The NIC responds by asserting the DEVSEL# signal and decoding the specific configuration register from the address bus and the byte enable signals.

Configuration consists of allocating system resources to the NIC and setting NIC-specific options. This is done by writing values into special PCI configuration registers, and into I/O registers. The location of this configuration space in the host processor's address map is system-dependent.

PCI configuration is performed by a BIOS routine supplied with the computer system. NIC-specific configuration is the driver's responsibility. For information on generating configuration cycles from driver software, see the PCI BIOS specification (available from the PCI SIG at the World Wide Web site www.pcisig.com.)

Table 13 summarizes the PCI configuration registers. Shaded spaces and all locations within the 256-byte configuration space that are not shown in the table are reserved.

Table 13 PCI Configuration Register Layout

Byte 3	Byte 2	Byte 1	Byte 0	Offset (Hex)
Data		PowerMgmtCtrl		E0
PowerMgmtCap		NextPtr	CapId	DC
				40–D8
MaxLat	MinGnt	InterruptPin	InterruptLine	3C
				38
			CapPtr	34
BiosRomControl				30
SubsystemId		SubsystemVendorId		2C
				28
				24
				20
				1C
				18
BaseAddress2 (memory)				14
BaseAddress1 (I/O)				10
	HeaderType	LatencyTimer	CacheLineSize	0C
ClassCode			RevisionId	08
PciStatus		PciCommand		04
DeviceId		VendorId		00

The following sections describe the PCI configuration registers.

VendorId

Synopsis	Contains the unique 16-bit manufacturer's ID allocated by the PCI SIG.
Type	Read-only
Size	16 bits
Offset	0

VendorId Register Format

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

The 3Com manufacturer ID is 10B7h.

DeviceId

Synopsis	Contains the 3Com-allocated 16-bit device ID for the NIC, which is read from EEPROM location 03h after reset.
Type	Read-only
Size	16 bits
Offset	2

DeviceId Register Format

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

PciCommand

Synopsis	Provides control over the NIC's ability to generate and respond to PCI cycles.
Type	Read/write
Size	16 bits
Offset	4

PciCommand Register Format

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

PciCommand Bit Descriptions

Bit	Name	Description
[0]	ioSpace	Allows the NIC to respond to I/O space accesses (if the NIC is in the D0 power state).
[1]	memorySpace	Allows the NIC to respond to memory accesses if the NIC is in the D0 power state.
[2]	busMaster	Allows NICs with bus master capability to initiate bus master cycles (if the NIC is in the D0 power state).
[4]	MWIEnable	Memory Write and Invalidate Enable. Allows the NIC to generate the MWI command.
[6]	parityErrorResponse	Controls how the NIC responds to parity errors. Setting this bit causes the NIC to take its normal action upon detecting a parity error. Clearing this bit causes the NIC to ignore parity errors. This bit is cleared upon system reset.
[8]	SERREnable	Enables the SERR# pin driver. A value of 0 disables the SERR# driver.

When a 0 is written to the PciCommand register, the NIC is logically disconnected from the PCI bus, except for configuration cycles.

PciStatus

Synopsis	Records status information for PCI bus events.
Type	Read/write
Size	16 bits
Offset	6

PciStatus Register Format

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

PciStatus Bit Descriptions

Bit	Name	Description
[4]	capabilitiesList	This read-only bit indicates the existence of a list of extended capabilities registers. The CapPtr register points to the start of the list.
[4]	udfSupported	This read-only bit indicates that the NIC supports the User Defined Fields format, as proposed by the PCI SIG.
[7]	fastBackToBack	This read-only bit indicates that the NIC, as a target, supports fast back-to-back transactions as defined in section 3.4.2 of the PCI specification, revision 2.0.
[8]	dataParityDetected	The NIC sets this bit when, as a master, it detects the PERR# signal asserted, and the parityErrorResponse bit is set in the PciCommand register.
[10:9]	devselTiming	This read-only field is used to encode the slowest time with which the NIC asserts the DEVSEL# signal. The NIC returns 2'b01, indicating support of "medium" speed DEVSEL# assertion.
[11]	signaledTargetAbort	The NIC asserts this bit when it terminates a bus transaction with target-abort.
[12]	receivedTargetAbort	The NIC asserts this bit when, operating as a bus master, its bus transaction is terminated with target-abort.
[13]	receivedMasterAbort	The NIC asserts this bit when, operating as a bus master, its bus transaction is terminated with master-abort.
[14]	signaledSystemError	This bit is set whenever the NIC asserts the SERR# signal.
[15]	detectedParityError	The NIC asserts this bit when it detects a parity error, regardless of whether parity error handling is enabled.

Although the PciStatus register is writable, write operations work in an unusual manner. Read/write bits in the register can be reset, but not set, by writing to PciStatus. A bit can be reset by writing a 1 to that bit position.

RevisionId

Synopsis	Provides a revision code for the PCI bridge ASIC.
Type	Read-only
Size	8 bits
Offset	8

RevisionId Register Format

7	6	5	4	3	2	1	0
0	0	1	0	0	0	0	1

The first version of the ASIC returns 21h. Succeeding versions are incremented.

ClassCode

Synopsis	Identifies the general function of the PCI device.
Type	Read-only
Size	24 bits
Offset	9

ClassCode Register Format

23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0

The NIC returns 020100h, indicating a token ring network controller.

CacheLineSize

Synopsis	Holds the system's cache line size, as written by the system BIOS.
Type	Read/write
Size	8 bits
Offset	C

CacheLineSize Register Format

7	6	5	4	3	2	1	0
0						0	0

The NIC uses the cache line size to optimize PCI bus master operation (choosing the best memory command, and so forth).

The value in the CacheLineSize register represents the number of dwords in a cache. CacheLineSize only supports powers of 2 from 4 to 64 (giving a range of 16 to 256 bytes). An unsupported value is treated the same as zero.

LatencyTimer

Synopsis	Specifies, in units of PCI bus clocks, the value of the latency timer for bus master operations.
Type	Read/write
Size	8 bits
Offset	D

LatencyTimer Register Format

7	6	5	4	3	2	1	0
					0	0	0

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

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

HeaderType

Synopsis	Identifies the NIC as a single-function PCI device, and specifies the configuration register layout shown in Table 13.
Type	Read-only
Size	8 bits
Offset	E

HeaderType Register Format

7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0

This field returns the value 00h.

BaseAddress1

Synopsis	Allows the system to define the I/O base address for the NIC's host register set.
Type	Read/write
Size	32 bits
Offset	10

PCI specifications require that base addresses be set as if the system used 32-bit addressing. The register returns 1 in bit 0 to indicate that this is an I/O base address (not a memory base address). The upper 25 bits of the register are writable, indicating that the NIC requires 128 bytes of I/O space in the system I/O map.

All host registers are mapped into both I/O and memory space.

BaseAddress1 Register Format

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

BaseAddress1 Bit Descriptions

Bit	Name	Description
[31:2]	ioBaseAddress	The system programs the I/O base address into this field. Because the NIC uses 128 bytes of I/O space, 25 bits are required to specify the base address. The hardware forces bits [6:2] to 0.
[1]	reservedByPci	This bit always reads 0.
[0]	ioSpaceIndicator	This read-only bit indicates that the base address specified is an I/O base address, not a memory base address.

BaseAddress2

Synopsis	Allows the system to define the memory base address for the NIC's host register set.
Type	Read/write
Size	32 bits
Offset	14

PCI specifications require that base addresses be set as if the system used 32-bit addressing. The BaseAddress2 register returns 0 in bit 0 to indicate that this is a memory base address (not an I/O base address). The upper 25 bits of the register are writable, indicating that the NIC requires 128 bytes of I/O space in the system I/O map.

All host registers are mapped into both I/O and memory space.

BaseAddress2 Register Format

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

BaseAddress2 Bit Descriptions

Bit	Name	Description
[31:4]	memBaseAddress	The system programs the memory base address into this field. Because the NIC uses 128 bytes of I/O space, 25 bits are required to specify the base address. The hardware forces bits [6:4] to 0.
[3]	prefetchable	This read-only bit is set to 0 to indicate that read operations from the 3C359 NIC have side effects and therefore are not prefetchable. Specifically, reading the IntStatusAuto register acknowledges interrupts and clears the IntStatus and InterruptEnable registers.
[2:1]	type	The value in this read-only field is determined by bit 2 of the PciParms1 register in EEPROM. When bit 2 of PciParms1 is zero, this field is 00b, indicating that the 3C359 NIC's register set may be mapped anywhere in the host's 32-bit memory space. When bit 2 of PciParms1 is one, this field is 01b, indicating that the 3C359 NIC's register set must be mapped into the first 1 MB of host memory space. The 3C359 NIC ships with this field set to 2'b00.
[0]	memSpaceIndicator	This read-only bit indicates that the base address specified is a memory base address, not an I/O base address.

SubsystemVendorId

Synopsis	Contains the two-byte subsystem vendor ID, which is read from EEPROM location 17h after system reset.
Type	Read-only
Size	16 bits
Offset	2C

SubsystemVendorId Register Format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Restored from EEPROM location 17h															

SubsystemId

Synopsis	Contains the two-byte subsystem ID, which is read from EEPROM location 18h after system reset.
Type	Read-only
Size	16 bits
Offset	2E

SubsystemId Register Format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Restored from EEPROM location 18h															

BiosRomControl

Synopsis	Allows the system to define the base address for the NIC's flash ROM.
Type	Read/write
Size	32 bits
Offset	30

BiosRomControl Register Format

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

BiosRomControl Bit Descriptions

Bit	Name	Description
[31:16]	romBaseAddress	The system programs the expansion ROM base address into this field. Because this field is 16 bits wide, the ROM is mapped to 64 KB boundaries.
[0]	addressDecodeEnable	When this bit is cleared, the NIC does not respond to flash ROM accesses. Setting this bit when the memorySpace bit in the PciCommand register is also set causes the NIC to respond to accesses in its configured expansion ROM space.

CapPtr

Synopsis	Points to the beginning of a chain of registers that describe enhanced functions.
Type	Read-only
Size	8 bits
Offset	34

CapPtr Register Format

7	6	5	4	3	2	1	0
1	1	0	1	1	1	0	0

CapPtr is a hard-coded value. This register returns DCh, which points to the power management registers.

InterruptLine

Synopsis	Communicates to the device driver the interrupt level that is being used for the device, which allows the driver to use the appropriate interrupt vector for its ISR.
Type	Read-only
Size	8 bits
Offset	3C

For 80x86 systems, the value in InterruptLine corresponds to the IRQ numbers (1 through 15) of the standard dual 8259 configuration, and the values 0 and 255 correspond to *disabled*.

InterruptLine Register Format

7	6	5	4	3	2	1	0

InterruptPin

Synopsis	Indicates which PCI interrupt pin the NIC uses.
Type	Read-only
Size	8 bits
Offset	3D

InterruptPin Register Format

7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	1

The 3C359 NIC always uses the INTA# interrupt pin, so 01h is returned.

MinGnt

Synopsis	Specifies, in 250-ns increments, how long a burst period the NIC requires when it is operating as a bus master.
Type	Read-only
Size	8 bits
Offset	3E

MinGnt Register Format

7	6	5	4	3	2	1	0
0	0	0	0	0	1	1	0

The system uses the value in the MinGnt register as a clue for setting the LatencyTimer register. The value for MinGnt is stored in the PciParms1 register in the EEPROM. The probable value for MinGnt is 6h, which implies a bus grant period of 1.5 microseconds.

MaxLat

Synopsis	Specifies, in 250-ns increments, how often the NIC requires the bus when it is operating as a bus master.
Type	Read-only
Size	8 bits
Offset	3F

MinGnt Register Format

7	6	5	4	3	2	1	0
0	0	1	0	0	0	0	0

The system uses the value in the MaxLat register as a clue for setting the LatencyTimer register. The value for MaxLat is stored in the PciParms1 register in the EEPROM. The value for MaxLat is 20h, which implies a latency tolerance of 8 microseconds.

Power Management Registers

This section describes the power management registers. For details on power management operation, see "Power Management" in Chapter 3.

CapId

Synopsis	Indicates the type of capability data structure.
Type	Read-only
Size	8 bits
Offset	DC

CapId Register Format

7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	1

CapId returns 01h to indicate a PCI power management structure.

NextPtr

Synopsis	Points to the next capability data structure in the capabilities list.
Type	Read-only
Size	8 bits
Offset	DD

NextPtr Register Format

7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	1

NextPtr returns 00h to indicate that there are no further data structures.

PowerMgmtCap

Synopsis	Provides information about the NIC's power management capabilities.
Type	Read-only
Size	16 bits
Offset	DE

PowerMgmtCap Register Format

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

PowerMgmtCap Bit Descriptions

Bit	Name	Description
[2:0]	version	This field returns 1h, as specified in the *PCI Bus Power Management Specification*.
[3]	pmeClock	This bit indicates whether or not a PCI clock is needed for the 3C359 NIC to assert the PME# signal. Because the 3C359 NIC does not need a clock, this bit is hard-coded to 0.
[4]	auxPower	This bit indicates whether or not an auxiliary power supply is required to support power management events in the D3cold state. The 3C359 NIC does not support this capability, so this bit is hard-coded to 0.
[8:5]		Reserved.
[9]	d1	This bit, when set, indicates that this device supports the D1 power state. This value of this bit is determined by bit 12 in the EEPROM PciParms1 word.
[10]	d2	This bit, when set, indicates that this device supports the D2 power state. This value of this bit is determined by bit 13 in the EEPROM PciParms1 word.
[15:11]	pmeSupport	This field indicates the power states from which this device is able to generate a power management event (assert the PME# signal). Each bit corresponds to a power state. A zero in a particular bit indicates that events cannot be generated from that state. This values of these bits are determined by bits [15:11] in the EEPROM PciParms1 word. The bits are defined as follows: ■ xxxx1 = Power management events possible from D0. ■ xxx1x = Power management events possible from D1. ■ xx1xx = Power management events possible from D2. ■ x1xxx = Power management events possible from $D3_{hot}$. ■ 1xxxx = Power management events possible from $D3_{cold}$.

The 3C359 NIC returns 00111b, which means that it can issue a power management event indication from states D0, D1, and D2.

PowerMgmtCtrl

Synopsis	Allows control over the power state and the power management interrupts.
Type	Read/write
Size	16 bits
Offset	E0

PowerMgmtCtrl Register Format

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

PowerMgmtCtrl Bit Descriptions

Bit	Name	Description
[1:0]	powerState	This read/write field is used to determine or set the NIC's power state. The following values are defined: ■ 00 = State D0, the full power on state. The NIC is fully operational. D0 can be followed by any other state (D1, D2, or $D3_{hot}$). ■ 01 = State D1, the low-power state. D1 is used when the host system turns off but the NIC remains on the ring to look for a wake-up packet. In D1 the NIC ignores all PCI activity except configuration accesses. The PCI clock continues to run. The NIC must be put into the wake-up packet monitoring mode before entering D1. In D1, the NIC is able to detect a wake-up packet and assert the PME# signal pin when it receives one. NIC context is saved in D1. D0 and D2 are the only states that can directly follow D1. To move the NIC from D1 to $D3_{hot}$, the system must first put the NIC into D0, close the NIC, and then move to $D3_{hot}$. ■ 10 = State D2, which is identical to D1 except that the PCI clock is shut off. D2 results in a power savings for the 3C359 NIC of about 1 watt, relative to D0 or D1. NIC context is saved in D2. D0 is the only state that can follow D2. To move the NIC from D2 to $D3_{hot}$, the system must first put the NIC into D0, close the NIC, and then move to $D3_{hot}$. ■ 11 = State $D3_{hot}$, the lowest power state. The NIC should be closed before entering $D3_{hot}$. In $D3_{hot}$, the RESETOUT signal to the MAC ASIC is asserted, the PWRDN signal is asserted to the PHY, and the 32-Mhz clock is gated off. All pins between the PCI bridge ASIC and the rest of the NIC are individually set to the appropriate state that minimizes leakage in the I/O circuitry. NIC context is saved in $D3_{hot}$. D0 is the only state that can follow $D3_{hot}$. When the transition to D0 is made, a GlobalReset command is automatically performed and autoinitialization is done (as if the RST# signal were asserted). If powerState is set to a nonzero value, the NIC does not respond to PCI I/O or memory cycles, and it cannot generate PCI bus master cycles.
[8]	pmeEn	When this read/write bit is set, the NIC is allowed to report wake-up events on the PME# signal. The default is 0.

PowerMgmtCtrl Bit Descriptions (continued)

Bit	Name	Description
[12:9]	dataSelect	This field selects the information that is reported in the Data register.
[14:13]	dataScale	This read-only field indicates the scaling factor for the information that is reported in the Data register. When the dataSelect bit is set to 0, 1, 2, 4, 5, or 6, 0b01 is returned, indicating that the power levels reported in Data are in units of 0.1 watts. When dataSelect is set to 3 or 7, 0b11 is returned, indicating that the power levels reported in Data are in units of 0.001 watts.
[15]	pmeStat	This bit is set when the NIC would normally assert the PME# signal, regardless of the state of the pmeEn bit. This bit drives the PME# signal, assuming that the pmeEn bit is set. Writing 1 to this bit clears it. Writing 0 (the default) has no effect.

(2 of 2)

Data

Synopsis	Provides information on the amount of power the 3C359 NIC consumes in different power states.
Type	Read-only
Size	8 bits
Offset	E3

Data Register Format

7	6	5	4	3	2	1	0
0	0						1

The information in the Data register is restored from the EEPROM PciParms2 register. The value read from this register depends on the setting of the dataSelect bit in the PowerMgmtCtrl register. See Table 14.

Table 14 Data Register Values Based on dataSelect Bit Settings

dataSelect Setting (PowerMgmtCtrl)	Interpretation	Value from PciParms2 (Hex)	Scale Factor (Watts)	Nominal Power Requirements (Watts)
0	D0 consumed	1F	0.1	3.1
1	D1 consumed	1F	0.1	3.1
2	D2 consumed	15	0.1	2.1
3	D3 consumed	11	0.001	0.017
4	D0 dissipated	1F	0.1	3.1
5	D1 dissipated	1F	0.1	3.1
6	D2 dissipated	15	0.1	2.1
7	D3 dissipated	11	0.001	0.17
8–15	Unsupported	00	N/A	N/A

5 EEPROM

This chapter provides information about the EEPROM contents and registers. The EEPROM is physically connected to the MAC ASIC. The host gains access to the EEPROM through the EeControl and EeData registers. After system reset, the MAC ASIC configures itself by reading configuration registers in EEPROM. Then the PCI bridge ASIC configures itself by reading other EEPROM configuration registers.

Data Format

Table 15 summarizes the contents of the 3C359 NIC EEPROM. The data fields are described in alphabetical order in the following sections.

Table 15 3C359 NIC EEPROM Contents

Address Offset (Hex)	Field Name	Value (Hex)	Read by:	
			MAC ASIC	PCI Bridge ASIC
00	3Com Node Address (word 0)	Variable		
01	3Com Node Address (word 1)	Variable		
02	3Com Node Address (word 2)	Variable		
03	DeviceId	3590	x	x
04	Manufacturing data - Date	Variable		
05	Manufacturing data - Division	0036		
06	Manufacturing data - Product Code	484C		
07	ManufacturerId	6D50		
08	SwitchSettings	3201	x	
09	ResourceRedirector	509F	x	
0A	ConfigurationControl	0000	x	
0B	Pmbar	F600*	x	x
0C	PciParms1	FC18		x
0D–0F	Reserved	0000		
10	OEM Node Address Word 0	Variable		
11	OEM Node Address Word 1	Variable		
12	OEM Node Address Word 2	Variable		
13-16	Reserved	0000		
17	SubsystemVendorId	Variable		x
18	SubsystemId	Variable		x
19	PciParms2	429E		x
1A–1E	Reserved	0000		
1F	Checksum	Variable		

* The Pmbar value is for the standard configuration, with no flash ROM installed. When a flash ROM is installed and remote program load (RPL) is enabled, the Pmbar value is F9C0Hh. When a flash ROM is installed and RPL is not enabled, the Pmbar value is F000h. For instructions on changing this value in EEPROM, see the documentation that comes with the flash ROM.

3Com Node Address

This field contains the 3Com node address for the NIC. This is *not* the address specified with the Open.NIC command; see "OEM Node Address" later in this chapter.

Checksum

The checksum for the EPPROM contents is a word-wide XOR computed across all words in EEPROM words zero through 1Eh, and written into word 1Fh.

ConfigurationControl

The MAC ASIC reads the NIC-specific configuration information in this field during configuration. This field contains no bits of interest to the driver.

ManufacturerId

This field contains 3Com's assigned EISA Manufacturer ID. It is a byte-swapped, encoded form of the string "TCM." This is included to aid software in identifying 3Com NICs in systems where a PCI BIOS is not available. This value has no significance in PCI operation (it is unrelated to the PCI VendorId value). It is not used by the NIC logic in any way.

Manufacturing Data

The manufacturing data fields are described below.

Date

This field contains the date of manufacture, encoded as follows:

Day	[4:0]: The day (1 through 31)
Month	[8:5]: The number of the month (1 through 12)
Year	[15:9]: The last two digits of the current year (0 through 99)

Division

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

Product Code

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

OEM Node Address

This is the address to be specified with the Open.NIC command. For 3Com NICs, this field contains the same value as in 3Com Node Address. OEM developers may choose to program this field with a different value.

The ordering of the bytes in the OEM Node Address field is important. OEM node address words are ordered in the EEPROM as shown in Table 16. Their order on the network is shown in Table 17.

Table 16 OEM Node Address Words in EEPROM

Byte	Byte	Offset
1	0	10
3	2	11
5	4	12

Table 17 OEM Node Address Words on the Network

Bytes	0	1	2	3	4	5
	Transmission order →					

The driver must read the OEM node address and insert it into the SA field of all downloaded frames. For more information on the ordering of bytes see "Byte Transmission Order" in Chapter 6. For more information on the SA field, see Appendix A.

PciParms1

The contents of this field are loaded into the ASIC to control various hardware functions related to PCI bus operation.

PciParms1 Bit Descriptions

Bit	Name	Description
[0]	fastBackToBack	Determines the value for the fastBackToBack bit in the PciStatus register.
[1]	udfSupported	Determines the value for the udfSupported bit in the PciStatus register.
[2]	lower1Meg	Provides the value for the type bit in the BaseAddress2 register.
[6:3]	minGnt	Determines the value returned in bits [4:1] of the MinGnt register.
[10:7]	maxLat	Determines the value returned in bits [5:2] of the MaxLat register.
[15:11]	pmeSupport	Provides the values for the following fields of the PowerMgmtCap register: pmeSupport, d1, and d2.

PciParms2

This field contains information on the NIC's power consumption. The operating system can access this information through the PCI configuration Data register.

PciParms2 Bit Descriptions

Bit	Name	Description
[5:1]	data1	These bits specify the NIC's power consumption in the D0 and D1 states. See "Data" in Chapter 4.
[10:6]	data2	These bits specify the NIC's power consumption in the D2 state. See "Data" in Chapter 4.
[15:11]	data3	These bits specify the NIC's power consumption in the D3 state. See "Data" in Chapter 4.

Pmbar

The MAC ASIC reads this field during configuration and makes its information available in the MAC ASIC register set. The Pmbar register contains bits to control NIC initialization and to enable writes to the flash memory.

DeviceId

This field contains the 2-byte device identifier, which is loaded into the PCI bridge ASIC and made available in the DeviceId register in the PCI configuration space.

ResourceRedirector The MAC ASIC reads the NIC-specific configuration information in this field during configuration. This field contains no bits of interest to the driver.

SubsystemId This is the 2-byte subsystem ID. 3Com NICs use the same code as the DeviceId data field. OEM developers may want to specify a different SubsystemId.

SubsystemVendorId This is the 2-byte subsystem vendor ID. 3Com NICs use the 3Com PCI vendor ID, 10B7h. OEM developers may want to specify a different SubsystemVendorId.

SwitchSettings The MAC ASIC reads the NIC-specific configuration information in this field during configuration. The only bit in this field that is useful to the driver is bit 1, ringSpeed, which is used when detecting ring speed as described in "Detecting Ring Speed" in Chapter 11.

MAC ASIC Registers

The following MAC ASIC registers are associated with the EEPROM.

EeControl

Synopsis	Provides access to the EEPROM.
Type	Read/write
Size	16 bits
Local address	1C8Ah

EeControl Register Format

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

EeControl Bit Descriptions

Bit	Name	Description
[15]	eeBusy	When this bit is set, the EEPROM is busy. Write operations are ignored and read data is not valid in the EeData register.
[7:6]	opcode	This field is used in conjunction with the address bit to specify EEPROM operations, as follows: ■ 10 = Read the data in the register specified by address. ■ 11 = Erase the register specified by address. ■ 01 = Write the register specified by address. ■ 00 = If address = 10XXXX, erase all registers. ■ 00 = If address = 01XXXX, write all registers with the data in the EeData register. ■ 00 = If address = 00XXXX, disable all write/erase modes. ■ 00 = If address = 11XXXX, enable all write/erase modes. This command must precede all write/erase commands.
[5:0]	address	See the opcode bit description above.

EeData

Synopsis	Contains the data to be written to EEPROM or read from EEPROM as specified by the EeControl register.
Type	Read/write
Size	16 bits
Local address	1C8Ch

EeData Register Format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Restored from EEPROM															

6 DOWNLOAD AND TRANSMISSION

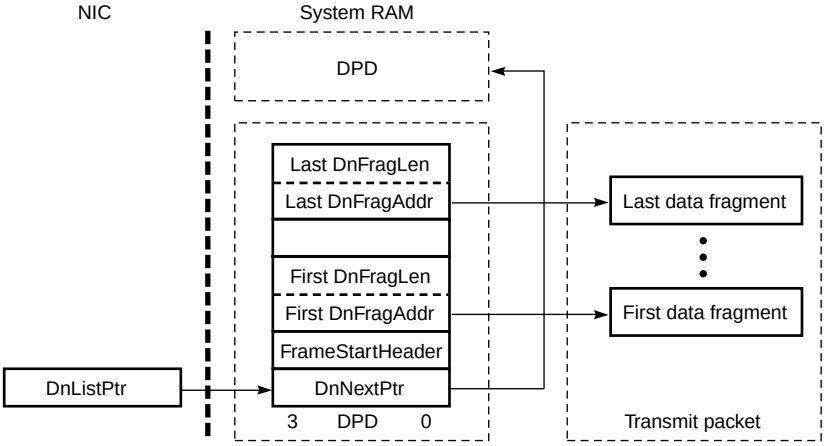
This chapter presents an overview of the packet download and transmission process, and defines the registers associated with the downloading and transmission of data.

3C359 NICs support a multipacket, multifragment gather process, whereby the driver builds descriptors that represent packets in system memory and links them together. The NIC follows the links, downloading multiple fragments per packet and generating interrupts when required.

Packet Download Model

A driver controls packet download by building a linked list of packet descriptors, called download packet descriptors (DPDs) when the network operating system issues a transmit request. The driver informs the NIC of a DPD's location and the NIC handles download and transmission of the packet with no further intervention from the driver. The linked list of DPDs, which is called the downlist, is illustrated in Figure 6.

Figure 6 Downlist



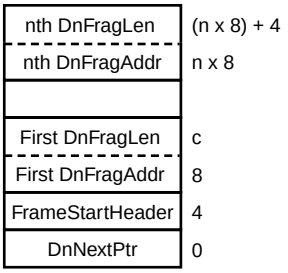
The driver places packets to be transmitted in data fragments (buffers) in system memory and then creates the list of DPDs that point to the fragments in system memory. All pointers are physical memory addresses (not virtual memory addresses).

The head of the list is the DPD that corresponds to the current download packet. The DnListPtr register points to this DPD. The NIC processes the DPD, fetching fragment address and fragment length values one at a time from the DPD and places them into NIC registers, which control the data download operations.

DPD Data Structure

The 3C359 NIC supports the DPD format shown in Figure 7.

Figure 7 DPD Format



DPD length ranges from 16 to 512 bytes. Each DPD describes up to 63 fragments, which consist of pairs of DnFragAddr and DnFragLen DPD entries.

DnNextPtr

The first dword in the DPD is the DnNextPtr entry, which contains the physical address of the next DPD in the downlist. If there are no more DPD entries in the downlist, then this value is zero.

DnNextPtr Format

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

DPDs must be aligned on 8-byte boundaries.

FrameStartHeader

The FrameStartHeader DPD entry (also called the FSH) contains packet control and status information.

FrameStartHeader Format

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

FrameStartHeader Bit Descriptions

Bit	Name	Description
[14:0]	frameLength	This field specifies the sum of all fragments that make up the transmit packet.
[15]	disableCrc	The driver sets this bit to inhibit the NIC from appending a CRC to the end of this packet. When disableCrc is set, it is expected that the packet's CRC would be supplied as part of the data downloaded to the FIFO. An exception to this occurs with a transmit underrun. In this case, either a guaranteed-bad CRC is appended to the packet (if so enabled by the firmware) or an abort delimiter is transmitted. When this bit is cleared, the NIC computes and appends CRCs for transmit packets.

FrameStartHeader Bit Descriptions (continued)

Bit	Name	Description
[16]	dnComplete	This bit indicates that the packet download is complete. The NIC sets this bit after it has finished downloading all of the fragments specified in the DPD, regardless of the setting of the dnIndicate bit.
[27]	txIndicate	When this bit is set, a txComplete interrupt occurs when the packet finishes transmitting. If this bit is cleared, no interrupt occurs unless a transmit error occurs.
[29]	dpdEmpty	This bit indicates that there is no packet data in this DPD, so the NIC should proceed directly to fetching the DnNextPtr DPD entry. A driver can use this feature to handle an empty DownList condition consistently. See "Polling on DnNextPtr" later in this chapter.
[30]	fshFormat	The driver must set this bit to 0 when the FSH is created.
[31]	dnIndicate	When this bit is set, a dnComplete interrupt occurs when a packet finishes downloading. If this bit is cleared, no interrupt occurs. The NIC reads this bit after the download operation has finished, allowing the host to change this bit while the download is in progress. The FSH is read twice: first to store the FSH bit values temporarily while the packet is being downloaded; and again after the download is finished to determine whether to generate an interrupt.

(2 of 2)

DnFragAddr

The DnFragAddr DPD entry contains the physical address of a contiguous block of data to be downloaded to the NIC and transmitted.

DnFragAddr Format

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

A fragment can start on any byte boundary. The hardware is responsible for aligning to dword and cache line boundaries.

DnFragLen

The DnFragLen DPD entry contains fragment length and control information for the block of data pointed to by the previous DnFragAddr DPD entry.

DnFragLen Format

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

DnFragLen Bit Descriptions

Bit	Name	Description
[14:0]	dnFragLen	This field contains the length of the contiguous block of data pointed to by the previous DnFragAddr DPD entry.
[31]	dnFragLast	The driver sets this bit to indicate that this is the last fragment of the transmit packet and that the NIC should proceed to the next DPD.

Packet Download

A packet download begins when all of the following conditions are true:

- The DnListPtr register is not equal to zero.
- The download engine is not in the DnStall command state.
- The transmitter has not experienced an underrun.
- If the downloadMode bit is set in the Config register, the available space in the download FIFO equals or exceeds the calculated burst size. If downloadMode is clear, the download FIFO must be empty.

Enabling Download

The NIC exits reset with the download engine in the idle state. Before attempting a download, the NIC must be opened (inserted in the ring) with the Open.NIC command. (For details on this and other software interface commands, see Chapter 11.)

After the NIC is opened, it is ready to start processing a downlist as soon as the driver writes a nonzero value into the DnListPtr register.

Simple Packet Download

The simplest example of packet download starts with the download engine idle, and an empty downlist, as is the case after reset. It is assumed that the NIC is already open.

To download a single packet, the driver creates a DPD with the addresses and lengths of the fragments containing the data to be transmitted. Since there are no more DPDs, the driver programs zero into the DnNextPtr DPD entry.

To start the download engine, the driver writes the address of the DPD into the DnListPtr register. The NIC proceeds to fetch information from the DPD and to move the packet data into the transmit FIFO.

When the download is finished, the NIC sets the appropriate status in the FrameStartHeader DPD entry and in the IntStatus register. It then reads the DnNextPtr DPD entry and places its value in the DnListPtr register. In the case of simple packet download, this is the only DPD entry in the downlist and the DnListPtr register is zero, which idles the download engine.

Polling on DnNextPtr

When the download engine fetches a DnNextPtr DPD entry that is equal to zero, the download goes idle. To restart the download, the DnListPtr register must be written with a nonzero value, which increases CPU utilization because it requires a bus transaction. However, the NIC can be programmed to poll on DnNextPtr automatically until the driver writes a nonzero value to it when it adds a DPD entry to the downlist. When the NIC reads a nonzero value, the download engine restarts with the DPD pointed to by the DnNextPtr entry it just fetched.

The DnPoll register controls this polling function. The value written to DnPoll determines the DnNextPtr polling interval. The polling function is enabled when DnPoll contains a nonzero value and the download engine is forced to read a DPD's DnNextPtr entry. This is done by creating a downlist that is not empty and pointing the NIC at it by writing the first DPD address to the DnListPtr register.

Sometimes it is desirable to start the polling process even though the downlist is empty, such as at initialization time. You can use the dpdEmpty bit in the FrameStartHeader DPD entry for this purpose. The procedure is:

1. Create a dummy DPD with its DnNextPtr entry set to 0 and the dpdEmpty bit of its FramestartHeader DPD entry set to 1. Because the dpdEmpty bit is set, the download engine does not process fragments.
2. Write a nonzero value to the DnPoll register.
3. Write the DnListPtr register with the address of the DPD created in step 1.

The NIC reads the FrameStartHeader entry in the DPD to which DnListPtr points, but because the dpdEmpty bit is set, the NIC immediately reads the DnNextPtr entry. Because DnNextPtr is zero at this time, the NIC begins polling on it. When the first packet is ready to be transmitted, the driver writes the address of its DPD into the dummy DPD's DnNextPtr entry. At the next polling cycle, the NIC sees the nonzero DnNextPtr entry and begins downloading as described in "Simple Packet Download" earlier.

It is possible for the driver to "short-circuit" the polling cycle after adding a DPD by simply reading DnListPtr. The NIC interprets this as a trigger to poll DnNextPtr. Polling is otherwise unaffected by this action.

Download Stalls and Idles

It is important to understand the distinction between stalling and idling the download engine.

A stall state is entered only when a DnStall command is issued. The stalled download engine must be restarted by issuing the DnUnStall command.

An idle state occurs if a DnReset or GlobalReset command has been issued, or if the DnListPtr register is 0. In these cases, simply writing a nonzero value to DnListPtr restarts the download. If the download is idle and also in the polling mode, then the download engine is restarted by writing a nonzero value to the DnNextPtr entry of the DPD that it is polling.

Download Completion

After downloading a packet, the NIC sets the dnComplete bit in the FrameStartHeader DPD entry.

The NIC can be configured to generate dnComplete interrupts when the downloading of packets has finished. These dnComplete interrupts can be generated on a per-packet basis by programming the appropriate value into the dnIndicate bit of each DPD's FrameStartHeader entry.

The NIC fetches the FrameStartHeader entry to examine the dnIndicate bit before packet download and again when the download is finished. This allows a driver to change dnIndicate while packet download is in progress. For example, a packet's DPD might be at the end of the downlist when it starts downloading, so the driver would set dnIndicate to generate an interrupt. However, if during the process of downloading this packet the driver were to add a new DPD to the end of the list, it might clear dnIndicate in the active DPD so that the interrupt is delayed until the next DPD finishes.

In response to a dnComplete interrupt, the driver acknowledges the interrupt and returns the DPD's buffers to the protocol. In the general case, in which the driver is

using a multipacket downlist, when the driver enters its interrupt handler, multiple packets may have been downloaded. To determine which packets in a list of DPDs have been downloaded, the driver can traverse the list, examining the dnComplete bit in each DPD.

Multipacket Lists

Generally, it is desirable to queue multiple DPDs. The driver links multiple DPDs together by pointing the DnNextPtr entry within each DPD at the next DPD, and programming zero into DnNextPtr in the last DPD.

Because the host and the NIC are frequently both accessing the downlist at the same time, the host must stall the NIC before modifying the downlist or writing a new value to the DnListPtr register (unless the value is already zero). Stalling the NIC ensures that the driver and the NIC do not process the same DPD at the same time, which causes erratic behavior.

Stalling is done by issuing a DnStall command. When DnStall is issued, the NIC finishes the current DPD but does not process the next DPD until the DnUnstall command is issued. When the host has finished manipulating the list, it issues DnUnStall.

Another way to restart the download process when it has been idled by a zero DnNextPtr value is by programming the NIC to poll automatically on the DnNextPtr DPD entry until a nonzero value has been written to it.

Priority Queueing

The DPD specifies a packet's location in host memory to the NIC. Multiple DPDs can be linked to construct a queue of transmit packets. The NIC downloads packets in the order that they appear in the queue.

The driver can change the order of the queue. Thus, if multiple low-priority packets are queued and a high-priority packet transmission is requested, the driver can place the high-priority packet near the head of the queue, just behind the DPD that the download engine is currently processing. It does this by stalling the download DMA engine with the DnStall command, linking the new high-priority packet to the head of the queue, and then restarting the DMA engine with the DnUnStall command. The DnStall command can be issued at any time, but it takes effect only on packet boundaries. The packet where the stall takes effect is made available to the driver in the DnListPtr register, so that the driver knows where to insert the high-priority packet.

The driver must also specify the priority of the packet in two 3-bit fields within the packet's AC and FC bytes. Together, these fields convey packet priority onto the ring and through bridges, switches, and other network equipment. For more information on these bytes, see "AC Field" and "FC Field" in Appendix A.

Adding DPDs to the End of the Downlist

You can add DPDs to the end of the downlist with polling disabled or enabled.

Polling Disabled The following sequence is recommended for adding a new DPD into the downlist when download polling is disabled (the DnPoll register is zero):

1. Stall the download engine by issuing the DnStall command.
2. Wait for DnStall to take effect by polling on the cmdInProgress bit in the IntStatus register. This bit typically clears immediately.
3. Update the DnNextPtr entry in the last DPD in the downlist to point at the new DPD.
4. Read the DnListPtr register.
5. If DnListPtr is zero, write the address of the new DPD to DnListPtr. (If DnListPtr is nonzero at the time the driver writes it, then the hardware ignores the write. If DnListPtr is zero, then the download has already reached the end of the downlist, as it was defined before adding the new DPD, and the hardware accepts the write.)
6. Unstall the download engine by issuing the DnUnStall command.

Polling Enabled When polling is enabled (the DnPoll register is nonzero), DPDs can be added with no register accesses. Point the last DPD's DnNextPtr entry at the new DPD.

Inserting a DPD Near the Head of the Downlist

To implement the priority queueing feature, you can insert packets into the downlist. Although DPDs cannot be added at the head of the downlist, they can be added just after the first active (unfinished) DPD.

Polling Disabled The following sequence is recommended for inserting a DPD near the head of the downlist when download polling is disabled (the DnPoll register is zero):

1. Stall the download engine by issuing the DnStall command.
2. Wait for DnStall to take effect by polling on the cmdInProgress bit in the IntStatus register. This bit typically clears immediately.
3. Read the DnListPtr register. The next action depends on whether DnListPtr is zero or nonzero:
   - If DnListPtr is zero, then the queue is empty and you can proceed with a simple packet download, as described in "Simple Packet Download" earlier in this chapter.
   - If DnListPtr is nonzero, then its value indicates which packet in the queue the download engine is processing. The download stalls when it is finished with this packet. It is safe for the driver to adjust the queue beginning with the next DPD in the queue, if there is one. Read the current DPD's DnNextPtr entry. The next action depends on whether DnNextPtr is zero or nonzero:
     - If DnNextPtr is zero, indicating that there is no following DPD, add the new DPD to the end of the downlist as described in "Adding DPDs to the End of the Downlist" earlier in this chapter.
     - If DnNextPtr is nonzero, check the priority of the packet just after the stall point. If it is not a priority packet, insert the new packet here. If it is a priority packet, then check the next packet. Continue checking packets until you find the last of the priority packets.

4 Set the DnNextPtr entry of the DPD just ahead of where the new packet is to be placed to point to the new DPD.

5 Set the DnNextPtr entry of the new DPD to point to the packet just following the newly inserted packet.

6 Write the address of the new DPD to DnListPtr. (If DnListPtr is nonzero at the time the driver writes it, then the hardware ignores the write. If DnListPtr is zero, then the download has already reached the end of the downlist, as it was defined before adding the new DPD, and the hardware accepts the write.)

7 Unstall the download engine by issuing the DnUnStall command.

Polling Enabled When polling is enabled (the DnPoll register is nonzero), DPDs can be inserted with no register accesses as follows:

1 Find the first nonpriority DPD that is not marked as downloaded (the dnComplete bit is zero).

2 Set this DPD's DnNextPtr entry to zero.

3 Check to see if this DPD is now marked as downloaded. If so, it is too late to insert a DPD at this DPD. Restore DnNextPtr, and move to the next DPD in the list and check again. If this DPD is not downloaded, go to the next step.

4 Point the inserted DPD's DnNextPtr where the first DPD once pointed.

5 Point the first DPD's DnNextPtr at the inserted DPD to complete the packet chain.

The driver must also specify the priority of the packet in the ppp field within the packet's AC byte and in the yyy field of the FC byte. For more information on these bytes, see "AC Field" and "FC Field" in Appendix A.

The ppp field enables the MAC to transmit the frame with a token of equal or lower priority than ppp. Upon transmission, the ppp field assumes the priority of the token used to transmit the frame. Thus, priority is lost when the frame is transmitted. To retain the priority, the yyy field in the FC byte is used.

Any software that forwards frames (a bridge, for example) should restore ppp with the value of yyy when forwarding frames, so that frame priority is retained through bridges, switches, and other networking equipment.

NIC Download Sequence The steps that the NIC performs for downloading packets are:

1 Check that the DnListPtr register is nonzero.

2 Check that the NIC is not in the DnStall command state.

3 Fetch the FrameStartHeader entry from the DPD to which the DnListPtr entry points. If the dpdEmpty bit is set, proceed directly to fetching the DnNextPtr entry. If dpdEmpty is clear, write the FrameStartHeader entry to the transmit FIFO.

4 Check that there is enough space in the download FIFO to support the calculated burst size.

5 One by one, fetch the DnFragAddr and DnFragLen entries from the DPD, and move the associated data fragments to the download FIFO.

6 After the last fragment download, set the dnComplete bit in the FrameStartHeader entry.

7 If the DnStall command was issued, wait until the DnUnStall command is issued.

8. Fetch the FrameStartHeader entry again. If the dnIndicate bit is set, set the dnComplete indication in the IntStatus register. This may, in turn, cause an interrupt, depending upon the masking of the bits in the IndicationEnable and InterruptEnable registers.
9. Fetch the DnNextPtr entry from the current DPD.
10. If the new DnNextPtr entry is zero, the download engine becomes idle. If polling is not enabled, the download engine waits until the driver writes a nonzero value to DnListPtr. If polling is enabled, then the current value of DnListPtr is retained while the NIC polls on the DnNextPtr DPD entry until it fetches a nonzero value, which is then written to DnListPtr.
11. Once a new DnListPtr has been acquired, return to step 2.
12. Repeat as necessary.

Byte Transmission Order

In different interfaces within the 3C359 NIC, the size of the data unit (bit, byte, word, or dword) varies. In the PCI interface, a data transaction on the bus may be one, two, three, or four bytes, depending on the size of the data fragment and its alignment. The 3C359 NIC packs all downloaded fragments together and reformats them into 2-byte words. Regardless of the alignment of the first fragment, the first word of the reformatted frame is a full word; that is, both bytes are transmitted.

Upon transmission, the reformatted frame is moved from the PCI bridge ASIC to the MAC ASIC as a stream of 2-byte words. The right-most byte is considered to be the most-significant byte, and is transmitted first, followed by the left-most byte. The last transfer of frame data between the two ASICs may be either a full word or a single byte, depending on whether the frame has an even or odd number of bytes.

The data fragments that make up a frame must contain all of these fields: AC, FC, DA, SA, and DATA. If the frame contains source route information, the fragments must also contain the RI field. The 3C359 NIC appends the SFD, FCS, EFD, and FS fields automatically. For more information on these fields, see Appendix A.

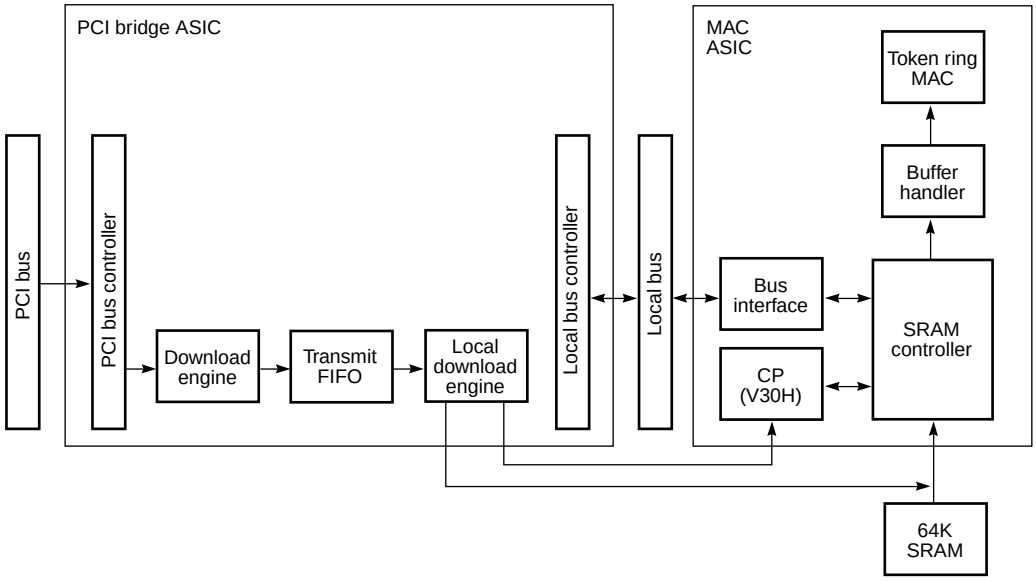
Packet Transmission

Once transmission is enabled, the NIC starts packet transmission as soon as either an entire packet has been downloaded to the transmit FIFO or the number of bytes in the FIFO is greater than the value in the TxStartThresh register.

Packet Transmission Model

Packet transmission is modeled as a logical FIFO. The PCI bridge ASIC downloads data from host memory to the transmit FIFO, a small but very fast burst FIFO. The download DMA engine places data into the transmit FIFO. The local download DMA engine supplies data from the FIFO to the MAC transmitter over the NIC's private memory bus, and constantly monitors for transmit underrun. See Figure 8.

Figure 8 Packet Transmission Path



The download engine tries to keep the FIFO full until the end of a packet has been downloaded, while also observing rules on burst size and frequency that are designed to optimize transfers across the PCI bus.

The downloadMode bit in the Config register controls the downloading of subsequent packets. Two modes allow a trade-off between performance and flexibility in queue reordering:

- In one mode, the download of a subsequent packet proceeds as soon as download of the current packet is finished, essentially keeping the FIFO full at all times. This mode offers the highest throughput.
- In the other mode, the download of a subsequent packet waits until the FIFO is empty. This mode offers the most flexibility for reordering the downlist.

Optionally, a completion indication may be given when the download or transmission of any frame is complete.

Optimized Packet Transmission

PCI bus latencies are generally very short, and bandwidth is very high. Thus, it is a good assumption that the host DMA can supply transmit data as fast as the transmitter needs it. Transmit packets should be queued as early as possible to maximize performance. To do this, the txStartThresh value should be kept very low (12 bytes is the practical minimum). If underruns occur at a rate that is considered excessive (for example, more than 1 in 5,000 frames) the driver should increase the value of txStartThresh. A threshold setting above 512 (the size of the download FIFO) results in a threshold of 512 (a full FIFO). Packets that are shorter than the value of txStartThresh are queued when they are completely downloaded. Conversely, if underruns occur at a very low rate, and the txStartThresh value is not already at its minimum, then the driver should lower the txStartThresh value.

You can achieve the best transmission throughput by setting the downloadMode bit in the Config register. This permits the download engine to process DPDs as long as there is room in the FIFO, regardless of whether or not other frames or

pieces of frames are also in the FIFO. The drawback to this mode is that when a priority frame is to be inserted into the queue, the FIFO may already be full of short frames. A priority frame cannot be placed ahead of these frames.

Reducing Interrupts

The transmit mechanism allows the driver software to perform optimizations that reduce the number of interrupts generated.

Limiting dnComplete Interrupts

The driver can limit the number of packets in the downlist for which a dnComplete interrupt is generated. It could, for example, only set the dnIndicate bit for the packet at the end of the list (clearing dnIndicate for the current tail before enqueueing each new packet). Or the driver might require an interrupt every *n* packets. In any case, on each interrupt the driver would then dequeue all of the packets that were downloaded before that interrupt occurred (DPDs in which the dnComplete bit is set or all packets in the list preceding the packet pointed to by DnListPtr). Obviously, there is a trade-off between latency and the number of interrupts taken. The driver writer is responsible for making this trade-off.

Using CountDown Timer Instead of dnComplete

The driver can mask off dnComplete interrupts and use the CountDown register to generate interrupts instead. The driver might look at the time it would take to transmit all the bytes currently in the transmit FIFO and queued in the downlist and set CountDown to half that time, for example. The driver would then use the intRequested interrupt (which results when CountDown expires) to dequeue all the DPDs in which the dnComplete bit is set or all the packets in front of DnListPtr. Again, this is just an example, and it is the driver writer's job to determine which algorithm to use.

Underrun Recovery

While the transmission is under way, the PCI bridge ASIC monitors the download FIFO for underrun. Underrun occurs if the MAC ASIC attempts to fetch transmit data, but the download FIFO is empty.

When an underrun occurs, the txUnderrun indication is set and the download data path is stopped. The host is responsible for cleaning up the PCI bridge ASIC's download data path, reenabling it, and requeueing the packet that underran. The underrunning packet can be identified by reading the DnListPtr register before issuing a DnReset command. The DnListPtr contains the pointer to the DPD that contains the packet that underran.

The procedure for underrun recovery follows:

1. The NIC suspends all download operations and retains the pointer to the underrunning DPD in the DnListPtr entry.
2. The host responds by reading the DnListPtr entry to locate the DPD of the underrunning packet.
3. The host issues a DnReset command to reset the underrun. This clears the dnComplete bit and the txUnderrun indication.
4. The host issues an AckInterrupt command with bit 0 set to clear the interruptLatch indication.

5 The host restores all transmit-related thresholds (the TxStartThresh register, in particular).

6 The host enables local download by issuing the DnEnable command and retransmits the packet by pointing the DnListPtr entry at the DPD of the underrunning frame.

Host Registers

The host registers that apply to download and transmission are described in the following sections.

DmaCtrl

Synopsis	Provides control and status information for bus master operations.
Type	Read/write
Size	32 bits
Offset	20

DmaCtrl is cleared by a reset.

DmaCtrl Register Format

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

DmaCtrl Bit Descriptions

Bit	Name	Description
[1]	dnCmplReq	This read-only bit is set to the value that the packet controller reads from the dnIndicate field in the FSH of the current DPD.
[2]	dnStalled	This read-only bit is set whenever downloading is stalled with the DnStall command. It is cleared by a DnUnStall command.
[3]	upComplete	This read-only bit is the same as upComplete in the IntStatus register, except that this bit is always visible regardless of the setting of the IndicationEnable mask. This bit is different from the upPktComplete bit in the UpPktStatus register in that upComplete latches on once an upComplete indication has occurred, whereas upPktComplete is cleared once an RxDiscard command is issued either by the driver or by hardware at upload completion. This bit is cleared by issuing an AckInterrupt command with the upCompleteAck bit set.
[4]	dnComplete	This read-only bit is the same as dnComplete in IntStatus, except that this bit is always visible regardless of the setting of the IndicationEnable mask. This bit is cleared by issuing an AckInterrupt command with the dnCompleteAck bit set.

(1 of 2)

DmaCtrl Bit Descriptions (continued)

Bit	Name	Description
[6]	armCountdown	This read-only bit specifies whether expiration of the Countdown register sets the intRequested bit. If armCountdown is clear, Countdown expiration does not set the intRequested bit. If armCountdown is set, expiration of Countdown sets intRequested. The armCountdown bit is cleared automatically by the act of setting intRequested, or when a zero value is written to Countdown. The armCountdown bit is set when a nonzero value is written to Countdown.
[7]	dnInProgress	This read-only bit indicates that a download operation is in progress. Drivers use this bit primarily in an underrun recovery routine. The driver waits for this bit to be cleared before issuing a DnReset command to clear the underrun condition. Before checking dnInProgress, issue a DnStall command to ensure that dnInProgress is not set as a result of the NIC being in a polling mode.
[8]	counterSpeed	This read/write bit sets the count rate for the Countdown and FreeTimer counters. When counterSpeed is cleared, the count rate is once every 8 microseconds (four byte times at 4 Mbps). When counterSpeed is set, the count rate is once every 2 microseconds (four byte times at 16 Mbps). By setting counterSpeed appropriately for the ring speed, conversions can be made between byte times and counter values using simple shift operations.
[9]	countdownMode	This read/write bit controls the operating mode of the Countdown register. With this bit cleared, Countdown begins its down counting operation as soon as a nonzero value is written to it. With this bit set, Countdown does not begin counting down until the dnComplete bit in the IntStatus register is set. For more information on the Countdown modes, see "Countdown" in Chapter 10.
[20]	defeatMWI	Setting this bit prevents the bus master logic from using the MWI PCI command.
[30]	targetAbort	This read-only bit is set when the NIC experiences a target abort sequence when operating as a bus master. This bit indicates a fatal error, and it must be cleared before further download or upload operations can proceed. This bit is cleared by issuing a GlobalReset command with the upDownReset mask bit cleared.
[31]	masterAbort	This read-only bit is set when the NIC experiences a master abort sequence when operating as a bus master. This bit indicates a fatal error, and it must be cleared before further download or upload operations can proceed. This bit is cleared by issuing a GlobalReset command with the upDownReset mask bit cleared.

(2 of 2)

DnBurstThresh

Synopsis	Defines a threshold that determines when bus master download requests are made.
Type	Read/write
Size	8 bits
Offset	41

DnBurstThresh Register Format

7	6	5	4	3	2	1	0
0	0	0	0				

The DnBurstThresh register determines when the NIC makes download bus master requests, based upon the available space in the transmit FIFO. The value in this register represents free space in the FIFO in units of 32 bytes. When the free space exceeds the threshold, the NIC can make a download request.

DnBurstThresh may be overridden by the DnPriReqThresh register mechanism. See "PCI Bus Master Operation" in Chapter 3 for information about the relationship between DnBurstThresh and DnPriReqThresh.

A value of zero is invalid. DnBurstThresh defaults to 8, a threshold of 256 bytes.

DnListPtr

Synopsis	Points to the current DPD in the downlist.
Type	Read/write
Size	32 bits
Offset	24

DnListPtr Register Format

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

The DnListPtr register holds the address of the current DPD in the downlist. The NIC interprets a value of zero in DnListPtr to mean that no more packets remain to be downloaded.

DnListPtr is cleared by reset.

DnListPtr can only point to addresses on 8-byte boundaries, so DPDs must be aligned on 8-byte boundaries.

DnListPtr may be written directly by host software to point the NIC at the head of a newly created downlist.

Writes to DnListPtr are ignored while the current value in the register is nonzero. To avoid access conflicts between the NIC and host software, the host must issue a DnStall command before writing to DnListPtr (unless the driver has specific knowledge that DnListPtr contains zero).

The NIC also updates DnListPtr while it processes DPDs in the downlist. As the NIC finishes processing a DPD, it fetches the value from the DnNextPtr entry. If the value is nonzero, the value is loaded into DnListPtr, allowing the download engine to move on to the next DPD.

If the NIC reads a value of zero from the current DPD, the current value in DnListPtr is preserved, and the download engine becomes idle. There are two ways the download engine can leave the idle state:

- The driver can write a nonzero value directly to DnListPtr.
- If polling is enabled, the download engine leaves the idle state when a nonzero value is finally fetched from DnNextPtr.

When the download engine is polling on DnNextPtr, a write of any value to DnListPtr stops the polling process. A read of DnListPtr truncates the polling interval (which results in an immediate poll cycle) and leaves polling mode active.

This register must always be written using a 4-byte write transaction. Anything less than four bytes could result in the download engine starting up with a transient DnListPtr.

DnPoll

Synopsis	Sets the poll rate of the DnNextPtr DPD entry.
Type	Read/write
Size	8 bits
Offset	2d

DnPoll Register Format

7	6	5	4	3	2	1	0
0							

The value in the DnPoll register determines the rate at which the DnNextPtr DPD entry is polled. Polling only occurs if DnPoll is nonzero and the value of the DnNextPtr entry fetched by the download engine at the end of DPD processing is zero.

Polling is disabled when DnPoll is cleared. DnPoll is cleared by hardware reset or setting the upDownReset bit in the GlobalReset command register and the dnReset bit in the DnReset command register.

The value in DnPoll represents 500-ns time intervals. The maximum value represents 63.5 µs.

DnPriReqThresh

Synopsis	Provides a threshold to set a point at which the download engine makes a priority bus master request.
Type	Read/write
Size	8 bits
Offset	2c

DnPriReqThresh Register Format

7	6	5	4	3	2	1	0
0	0	0	0				

The value in the DnPriReqThresh register sets a point at which the download engine makes a priority bus master request to the bus master arbiter. A priority download request has priority over the upload engine. When the amount of data in the download FIFO falls below the value implied by DnPriReqThresh, the priority bus request is made.

The value in DnPriReqThresh represents data in the transmit FIFO in terms of 32-byte portions. For example, a DnPriReqThresh value of 4 causes the NIC to request the bus when the amount of data in the FIFO falls below 128 bytes.

DnPriReqThresh resets to 4, or a threshold of 128 bytes. DnPriReqThresh is cleared by hardware reset or setting the upDownReset bit in the GlobalReset command register and the dnReset bit in the DnReset command register.

TxStartThresh

Synopsis	Provides for an early transmission start based upon the number of packet bytes downloaded to the NIC.
Type	Read-only
Size	16 bits
Offset	58

TxStartThresh Register Format

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

The value in the TxStartThresh register is used to control early packet transmission. Packet queueing to the transmitter begins when the number of bytes for the packet resident on the NIC is greater than the value implied by TxStartThresh.

TxStartThresh is set with the SetTxStartThresh command.

The recommended minimum threshold is 256 bytes.

TxStartThresh resets to 8,188 bytes, which disables the threshold mechanism. Thresholds above 512 truncate to 512. To correctly program the start threshold, the numerical value for the desired threshold must be right-shifted two bits. This value is then used in the SetTxStartThresh command.

7 RECEPTION AND UPLOAD

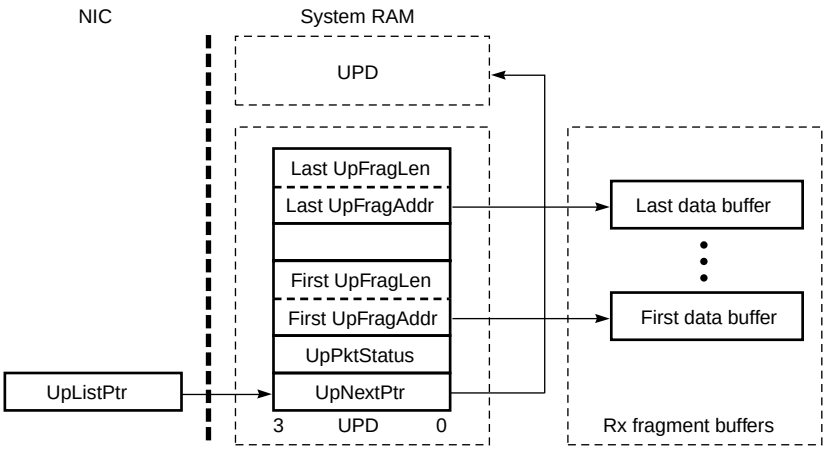
This chapter presents an overview of the packet reception and uploading process and defines the registers associated with the reception and uploading of data.

Packet Upload Model

The NIC supports a multipacket, multifragment scatter process, whereby incoming packets are moved to system memory buffers defined by descriptors. The descriptors themselves also reside in system memory and are linked together by the host CPU.

The packet upload model is similar to the download model. Upload is structured around a linked list of packet descriptors, called upload packet descriptors (UPDs). UPDs contain pointers to the fragment buffers into which the NIC is to place receive data. The linked list of UPDs, called the uplist, is illustrated in Figure 9.

Figure 9 Uplist

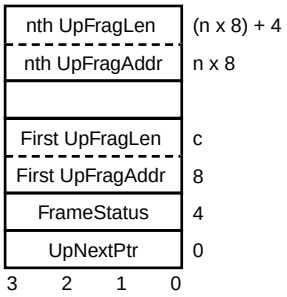


The head of the uplist is the UPD that corresponds to the current upload packet. The UpListPtr register points to this UPD. As the UPD is processed, the NIC fetches fragment address and fragment length values one at a time from the UPD and places them into NIC registers, which control the data upload operations.

UPD Data Structure

A UPD is 16 to 512 bytes long. It contains the UpNextPtr and FrameStatus entries, and from 1 to 63 pairs of UpFragAddr and UpFragLen entries. See Figure 10.

Figure 10 UPD Format



UpNextPtr The first dword in the UPD is the UpNextPtr entry, which contains the physical address of the next UPD in the uplist. If this is the last UPD in the uplist, then this value is zero.

UpNextPtr Format

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

UPDs must be aligned on 16-byte physical address boundaries.

FrameStatus The second dword in the UPD is the FrameStatus entry. At the end of packet upload, the NIC writes the value of the FrameStatus entry into this location in the UPD. Only the updFull and updComplete bits are valid for every FrameStatus in the uplist. All other bits are valid only if the upComplete bit is set and the rxOverrun bit is clear in the IntStatus register.

FrameStatus Format

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

FrameStatus Bit Descriptions

Bit	Name	Description
[14:0]	upPktLength	This field indicates the length of the uploaded packet when the updComplete bit is set.
[18:16]	arrMatch	This field indicates the Address Recognition register (ARR) that matched the destination address of the received packet. Only the following codes are used: ■ 000 = ARR0 (this node address) ■ 100 = ARR4 (this node address, MAC buffers) ■ 101 = ARR5 (used for source routing) ■ 111 = ARR7 (used for functional addressing)

FrameStatus Bit Descriptions (continued)

Bit	Name	Description
[19]	rxOverrun	Indicates a packet that has overrun. No status other than upComplete is valid if the packet overran. The driver should discard overrunning packets. The rxOverrun bit is set under either of these conditions: ■ The number of available buffers in NIC memory was insufficient to hold the complete frame. ■ The frame exceeded the limit specified by the SetMaxBytes command. When rxOverrun is set, the rxComplete bit is also set.
[21]	rxFc	This is the frame copied bit (fc) of the received FS field. See Appendix A.
[22]	rxAr	This is the address recognized bit (ar) of the received FS field. See Appendix A.
[23]	updComplete	Upload of the frame in this UPD is complete. When the updComplete bit is set, the updFull bit is never set, even if the packet fully fills the UPD.
[24]	updFull	When set, this bit indicates that the packet overflowed the UPD. That is, the fragments specified in the UPD used to upload the frame were too small to fit the entire frame. No other status is valid if the updFull bit is set. This is not a fatal condition. When updFull occurs, the upload continues to the next UPD, if there is one, or goes idle if there is none. If idled, the upload can be resumed after the driver provides another UPD and informs the NIC of such. See "Early Interrupts" later in this chapter for more information.
[26:25]	receiveStatusCode	This field provides encoded receive status as follows: ■ 00 = No error (normal completion) ■ 01 = An aborted packet was received ■ 10 = An implicit abort was received for one of the following reasons: The MAC detected a BURST4 condition. The -REDY signal was lost from the PHY. No EFD was received. ■ 11 = Either the ar bit pair or the fc bit pair in the FS field did not match.
[27]	sourceRouteCompare	When set, this bit indicates that the received packet passed the source route filter.
[28]	redi	(Returned error detect indication.) When set, this bit indicates that a remote station has detected a CRC or code violation in the packet. It is the edi bit of the EFD field (see "SFD and EFD Fields" in Appendix A).
[29]	rlpedi	(Receive local packet error detect indication.) When set, this bit indicates that the receiver detected a CRC or code violation while it was receiving the packet.
[30]	groupAddress	When set, this bit indicates that the received packet has a group address.
[31]	broadcastAddress	When set, this bit indicates that the received packet has a broadcast address.

(2 of 2)

UpFragAddr The third (fifth, and so on) dword in the UPD is the UpFragAddr entry, which contains the physical address of a contiguous block of system memory to which receive data is to be uploaded.

UpFragAddr Format

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

A fragment can start on any byte boundary. The hardware is responsible for aligning to dword or cache line boundaries, or both.

UpFragLen The fourth (sixth, and so on) dword in the UPD is the UpFragLen entry, which contains fragment length and control information for the block of data pointed to by the previous UpFragAddr UPD entry.

UpFragLen Format

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

UpFragLen Bit Descriptions

Bit	Name	Description
[14:0]	upFragLen	This field contains the length of the contiguous block of data pointed to by the previous UpFragAddr UPD entry.
[31]	upLastFrag	This is the last fragment defined by the current UPD. If the frame has not been fully uploaded by the time this fragment is filled, an updFull is indicated in the UPD's FrameStatus. Upload of the packet continues into the next UPD.

Packet Reception

The following sections describe various aspects of packet reception.

Enabling Reception

The NIC exits reset with the upload engine in the idle state. Reception is enabled when the NIC is opened (inserted in the ring) with the Open.Nic command. (For details on this and other software interface commands, see Chapter 11.) Once reception is enabled, packets are received according to the address filter specified in the Open.Nic or Modify.Open.Parms commands.

After the NIC is opened, it is ready to start processing an uplist as soon as the driver writes a nonzero value into the UpListPtr register.

Packet Reception Model

Packet reception is modeled as a logical FIFO. The network interface places data that is received into the FIFO. The system interface removes data from the FIFO.

The FIFO is composed of these entities:

- A burst FIFO handles high-speed bursts on the PCI bus. The burst FIFO size is 64 dwords by 32 bits. It can supply 32-bit words to the PCI bus at 33 MHz.
- A much larger but slower FIFO is implemented off-chip in private SRAM. This external FIFO is slower, but very deep (roughly 35 KB).

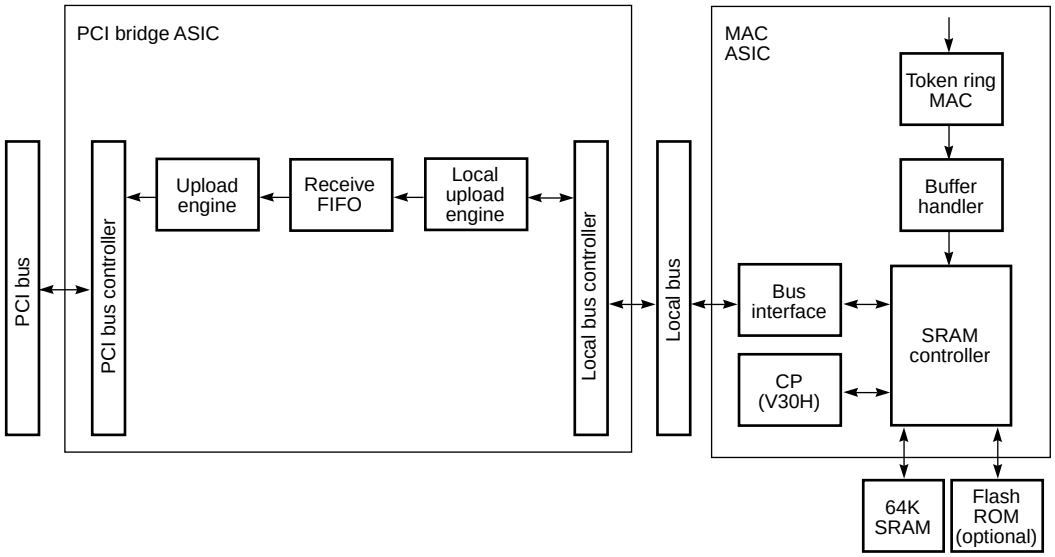
The burst FIFO can be replenished from the external FIFO at roughly 8 MB per second, allowing rapid back-to-back bursts on PCI.

The packet data placed in the FIFO consists of the AC physical control field through the source routing information field (RI), and can include the frame check sequence field (FCS). For details on these fields, see Appendix A.

Receive data is moved only by bus mastering DMA. There is no provision for using slave accesses to move data.

As shown in Figure 11, the MAC receiver places receive data into a series of buffers in private memory. The PCI bridge ASIC's local upload engine retrieves data from these buffers and places it in the upload FIFO on the PCI bridge ASIC. The local upload engine extracts each frame's status and length from the data stream, which it makes available to the driver in a hardware register. The host upload engine bursts data out of the FIFO into host buffers, as described by the UPDs, while observing the same set of rules for burst frequency and size as the download engine.

Figure 11 Packet Reception Path



Packet Upload

Any operation that the host performs on receive packets is performed on the top packet only. The top packet is the oldest packet in the FIFO, and the one that the upload DMA engine is either transferring or is ready to transfer next. It is also the packet for which status is made visible to the host.

In the 3C359 NIC, the top packet becomes *visible*—and thus eligible for uploading—after 64 bytes have been received, or packet reception is completed, whichever comes first.

Upload Modes

To provide optimal performance for different software environments, upload can operate in the standard store-and-forward mode or in parallel tasking mode, as described in "Store-and-Forward Packet Reception" and "Parallel Tasking Packet Reception" later in this chapter. It is possible to use a combination of the two modes when the Get_RCB-1 call can be used, such as in a Novell environment

under DOS. This approach has the high throughput of Parallel Tasking® technology, but because a second copy of data is required (from the RCB-1 buffers to the final application buffers), CPU utilization is compromised. This combined mode is described in "Combining Packet Reception Modes" later in this chapter.

Simple Packet Upload

The simplest example of packet upload starts with the upload engine idle and an empty uplist, as is the case after a reset. To upload a packet, the driver creates a UPD with the addresses and lengths of the buffers to be used (typically one buffer, equal to the maximum packet size). Because there are no more UPDs, the driver programs zero into the UpNextPtr UPD entry.

Next, the driver writes the address of the UPD into the UpListPtr register. Assuming there is a receive packet in the FIFO, the NIC proceeds to fetch information from the UPD and move the packet data into the buffers.

With receives, the driver probably needs to set up one or more UPDs and their associated buffers before reception of a packet. One approach is simply to allocate a block of full-size packet buffers in its own data space and create UPDs that point to the buffers, as described in "Store-and-Forward Packet Reception"later in this chapter. Data is subsequently copied out of these temporary buffers into protocol buffers after reception and upload have been completed. Another approach is for the driver to request the buffers from the protocol ahead of time, as described in "Parallel Tasking Packet Reception" later in this chapter.

Similar to the download DnStall and DnUnStall commands, which stall the download engine, there are UpStall and UpUnStall commands to stall the upload engine. The driver should issue an UpStall command before modifying the list pointers in the uplist. As with download stalls, upload stalls take effect at packet descriptor boundaries. However, unlike a DPD, a UPD boundary may be in the middle of a packet, because upload can use multiple UPDs per packet, whereas download uses only a single DPD per packet. Thus, when modifying the uplist, be aware that the stall may occur in the middle of a packet.

As with download, the upload engine becomes idle if it fetches an UpListPtr register value of zero. It also stalls (unless polling is enabled) if the updComplete or updFull bit in the current UPD's FrameStatus entry is set.

Packet Upload Completion

Normally, for every top frame, an rxComplete indication is given at the end of reception, followed by an upComplete indication at the end of upload. At the time upComplete is set, hardware issues an internal RxDiscard command, which clears rxComplete. Thus rxComplete and upComplete are mutually exclusive.

The rxComplete indication disappears when upComplete is set. Therefore, to prevent spurious interrupts, upComplete should only be disabled if rxComplete is also disabled.

The rxComplete indication is not followed by upComplete when the driver issues an RxDiscard command before the upComplete is given.

When there is more than one complete frame in NIC memory, rxComplete is cleared after RxDiscard and then set again to indicate that the new top frame is complete. But although the setting of rxComplete happens very quickly following the discard, the associated status in the UpPktStatus register is delayed because

the hardware must first collect the new top frame's status and length from local buffers. The time required to do this varies according to frame length. The upper bound is about 20 microseconds when hashing is not enabled.

Setting rxComplete immediately instead of waiting for status and length to be recovered provides the driver with a quick indication of whether or not another complete frame has been received. Thus, the driver is able to process the frame without the need to exit and return to its interrupt service routine (ISR).

If hash filtering is enabled, the new top frame may not pass the hash filter. In this case, the PCI bridge ASIC discards the frame automatically, without issuing an rxComplete. The next frame then becomes the new top frame and the process repeats. An rxComplete is issued only after a frame passes the hash filter. So, with hash filtering enabled, the time until the next rxComplete is indeterminate, because it depends on the number of frames that fail to pass the filter. For more on hash filtering, see "Multicast Filtering" later in this chapter.

Store-and-Forward Packet Reception

The standard reception process is a store-and-forward mode characterized by lower performance than optimized reception, but which allows better CPU utilization. In some environments, store-and-forward is the preferred operating mode. Store-and-forward mode saves on host memory usage because large temporary buffers are not used for frame data. This mode also provides the lowest CPU utilization metric, because there is no double copying of frames. However, this mode is not suggested for servers, because of the 3C359 NIC's limited on-card memory.

For standard packet reception, a *lookahead* UPD that is large enough to hold only a packet header is predefined in driver memory. The UpNextPtr entry should point back to this UPD. Thus, the UPD structure is a ring that contains only one UPD. This UPD might be 128 or 256 bytes long. The updNeeded interrupt should be disabled, and the rxComplete and upComplete interrupts enabled.

Store-and-forward operating mode is very efficient because the packet is copied only once to host memory (except for that portion of the lookahead UPD that is copied twice), or not at all if the packet is of no interest. The entire packet is received and stored in NIC memory. The lookahead UPD, its status, and its size are determined, and then it is copied (forwarded) to protocol buffers. This mode should be used when CPU utilization is to be kept to a minimum. (See "Parallel Tasking Packet Reception" later in this chapter for a contrasting operating mode.)

Store-and-Forward Procedure

The steps for store-and-forward packet reception are:

1. When the NIC begins to receive a frame, it uses the predefined UPD to upload the frame. If the frame is larger than the UPD, the upload stops when the UPD is full, but the NIC continues to receive and buffer the frame.
2. When the NIC has received the entire frame, it issues an rxComplete indication.
3. A short time after the rxComplete indication (from about 1 microsecond for short frames to 20 microseconds for 18 KB frames), hardware recovers the frame status and size and makes them available in the UpPktStatus register. Between the time when the rxComplete indication is issued and the UpPktStatus register is valid, bits [31:15] (the status field) of UpPktStatus are zero. The driver, in response to

rxComplete, should poll on the status field bits for a nonzero value before moving on.

4 The driver, after examining the UpPktStatus register and the frame header in the UPD, takes one of the following actions:

- If the frame is of no interest, discards it by issuing the RxDiscard command. This purges the packet from its FIFO, acknowledges the rxComplete indication, and makes the next packet in line (if any) the new top packet. The driver should also recover the UPD if the frame was discarded.
- If the frame is of interest, masks off the rxComplete indication by clearing the corresponding bit in the InterruptEnable register, and requests buffers from the upper layers, into which the frame is to be copied. When the list of buffers is made available and organized by the driver into a UPD, the driver unstalls the NIC. The NIC then proceeds to upload the remainder of the packet using the new UPD. Meanwhile, the driver needs to copy that portion of the frame that resides in the original UPD and is of interest, into the protocol buffer. See "Multipacket Lists" later in this chapter for the recommended method to unstall the upload.

5 When the upload is finished, the upComplete indication is issued and the NIC issues an internal RxDiscard command, which makes the next frame in line the top frame.

Minimizing Register Accesses

Because register accesses are very expensive in a PCI environment, the driver should strive to minimize them in store-and-forward mode.

The following method for minimizing register accesses requires, for each frame, two interrupts (rxComplete and upComplete), two reads of the IntStatus register (to determine the interrupt source), two acknowledgments of the interrupts (register writes), and one read of the UpPktStatus register. It may also be necessary to read the UpListPtr register to guarantee that the driver knows which UPD the NIC is polling on. The method is:

1 Organize two or more UPDs into a ring. Each UPD should be only large enough to hold the packet header.

2 Clear the updComplete and updFull bits of one UPD's FrameStatus entry and set either the updComplete or updFull bit of the other UPD. When the updComplete and updFull bits are clear, the UPD is available for upload, but when either is set, the UPD is considered used and unavailable to the hardware.

3 When the bits are cleared, write the location of the UPD to the NIC's UpListPtr register. Also, write the UpPoll register to set the desired poll interval.

When the NIC begins to receive a frame, it uploads and fills the UPD (assuming the frame is bigger than the UPD). The NIC fills out the FrameStatus entry of this UPD (with its updFull bit set) and moves to the next UPD. However, because the updComplete or updFull bits of the second UPD are set, the upload pauses while the NIC monitors both of these bits continuously at the interval specified in the UpPoll register, looking for them to be clear.

4 When frame reception is completed, the rxComplete indication is issued. The driver clears the rxComplete interrupt enable bit and requests buffers from the protocol. After getting these buffers from the upper layers, the driver fills out the

UPD that the NIC is pointing to with these buffers. Then the driver clears the updComplete and updFull bits in the UPD's FrameStatus entry.

The NIC, which continuously polls on the updComplete and updFull bits at the poll interval, sees that they are cleared and resumes the upload.

5 When the upload is finished, the NIC, as always, sets the upComplete indication, writes the UpPktStatus register to the UPD's FrameStatus entry, issues an internal RxDiscard command, fetches the UpNextPtr entry from the UPD and loads it into UpListPtr, and fetches the FrameStatus entry from the new UPD. Since the updFull bit in FrameStatus is set (from step 2), an implicit stall is done.

6 The driver, after processing the earlier frame, clears the updFull bit of the first UPD so that it can be reused for the next frame. The upload unstalls itself when it sees that updFull bit is clear.

7 Repeat as necessary.

Parallel Tasking Packet Reception

Instead of accumulating packets in NIC memory, the driver can preallocate buffers (predefine UPDs) in driver memory space and let the NIC upload packets into these buffers as they are received. This process is called Parallel Tasking technology. At the end of the upload, frame length and status are deposited in the FrameStatus UPD entry and an upComplete indication is given. After the driver gets buffers from the protocol, it copies the frame into the buffers using a memory-to-memory copy.

With Parallel Tasking technology, when the upComplete indication occurs, the packet is already in host memory. However, the packet still needs to be placed in protocol buffers, so it must be copied from the driver's buffers to the protocol's buffers. This memory-to-memory copy takes place very quickly in some processors.

Parallel Tasking technology consumes more CPU time than store-and-forward mode, because the packet must be copied twice. However, performance is better because memory-to-memory copies are faster than copies across the PCI bus. Parallel Tasking technology also increases the size of the driver, because buffers are required in the data space. Because it requires a double copy of data, Parallel Tasking technology sacrifices some CPU utilization to increase throughput. This technology is useful if driver size is not a great concern, such as might be the case with drivers that can use extended memory.

Combining Packet Reception Modes

It is possible to combine store-and-forward and parallel tasking modes of packet reception. In this combined mode, the driver uses store-and-forward mode to receive small frames that fit completely into a lookahead buffer. It uses parallel tasking to receive additional frames into more lookahead buffers while the last-completed lookahead buffer is still being processed.

The driver organizes at least two UPDs into a ring, so that each UPD points to another. Each UPD is large enough to hold a small frame that occupies perhaps 512 bytes. The FrameStatus field in one of the UPDs is cleared, while the updFull bit is set in the other UPD's FrameStatus field. If all incoming frames fit into the memory provided by each UPD, this configuration basically uses parallel tasking mode to receive all data. However, if larger frames are also received, this configuration uses store-and-forward mode as well, because the driver sees only a portion of the total incoming frame when it receives an interrupt.

The driver must enable both the upComplete and updNeeded interrupts to use this mode. The updNeeded interrupt occurs whenever the NIC has transferred a portion of a large frame into one of the lookahead buffers, or anytime the NIC starts trying to upload data into another lookahead buffer before the driver has finished processing the one pointed to by the new UPD. The upComplete interrupt occurs whenever a frame is completely received into a lookahead buffer, or after the driver has provided additional host memory in response to the updNeeded interrupt.

The steps for using store-and-forward and parallel tasking modes simultaneously are:

1. When the NIC begins to receive a frame, it uses the current UPD to upload the frame. If the frame fits completely within the space provided by the UPD, the NIC generates an upComplete interrupt. If it does not fit, the NIC generates the updNeeded interrupt but continues to receive and buffer the rest of the frame in its local memory.
2. When upComplete occurs, the driver immediately clears the updFull bit in the adjacent UPD's FrameStatus field. This allows the NIC to start transferring a new frame into that UPD when one arrives. The driver checks the size of the frame to determine whether the entire frame has been received. If so, the driver handles the frame using store-and-forward techniques. Otherwise, the upComplete interrupt must have been generated following an earlier updNeeded interrupt, and because of the way that interrupt is handled, the frame has already been copied into host memory.
3. When the updNeeded interrupt occurs, the driver must provide enough host memory so that the NIC can finish uploading the remainder of the incoming frame. The driver shows the first portion of the frame to the protocol. If the protocol wants to receive the frame, the driver fills out the current UPD with pointers to the memory fragments provided by the protocol. Otherwise, the driver discards the frame using the RxDiscard command.

Multicast Filtering

The 3C359 NIC uses a hashing technique to filter multicast packets during reception. If a frame fails to pass the hash filter, the upload engine issues an RxDiscard command without notifying the host of the frame's presence. If the frame passes the hash, the frame is uploaded. The NIC must be set to promiscuous group receive mode when hashing is enabled to allow the NIC to receive multicast (group) addressed frames. To enable promiscuous group receive mode, use the procedure in "Multicast Reception" in Chapter 11.

The least-significant six bits of the result are used as an index into the hash table. If the indexed hash table entry is set to 1, the frame is uploaded. If the entry is 0, the frame is discarded with no indication given to the host.

Because hashing is not a perfect filter—multiple distinct destination addresses can produce the same result after the hash operation has been applied to them—the driver must also filter the address before it decides to accept a multicast frame. A multicast frame that has passed the hash filter has the groupAddress bit set in the UpPktStatus register and the FrameStatus UPD entry.

Hashing is performed by accumulating a CRC on the frame's destination address if the destination address's most-significant bit is a 1, which indicates a group frame. The polynomial used for the CRC calculation is:

$$g(X) = X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^8 + X^7 + X^5 + X^4 + X^2 + X^1 + X^0$$

No part of a frame is uploaded before the result of the hash is known.

Hashing is enabled with the Command register.

Multipacket Lists

Sometimes it is desirable for the driver to create a list of multiple UPDs. Multiple UPDs are linked together by pointing the UpNextPtr entry within each UPD at the next UPD and programming zero into UpNextPtr in the last UPD.

One upload option that differs from download is that the uplist can be formed into a ring by writing the address of the first UPD, instead of zero, into the UpNextPtr entry of the last UPD. The NIC issues an internal UpStall command if it fetches an UpListPtr register value for a UPD that has already been used (that is, one in which either the updComplete or updFull bit is set in the FrameStatus entry). When the driver finishes processing a UPD, it should leave the FrameStatus entry cleared and issue an UpUnStall command, just in case the NIC has already read FrameStatus and stalled.

The following sequence is recommended for adding UPDs to the uplist:

1. Stall the upload engine by issuing the UpStall command.
2. Update the UpNextPtr entry in the last UPD in the uplist to point at the "new" UPD.
3. Read the UpListPtr UPD register.
4. If UpListPtr was zero, write the address of the new UPD into UpListPtr. (If the entry was nonzero, do nothing.)
5. Unstall the upload engine by issuing the UpUnStall command.

Using Multiple UPDs

It is possible to use more than one UPD to buffer a packet. Essentially, if an upload fills all the fragments of a UPD, the upload simply indicates updFull in that UPD's FrameStatus entry and continues the upload using the next UPD. This can go on as long as there are UPDs and until the frame ends. The FrameStatus of the UPD that holds the end of the frame is updated with final frame status and length. The updFull indication is not set in this UPD, but the updComplete indication is.

This capability is helpful for receiving packets as described in "Store-and-Forward Packet Reception" earlier in this chapter. It is also useful for parallel tasking when you do not want to preallocate a number of UPDs, each of which is big enough to hold the maximum frame size.

Early Interrupts

The 3C359 NIC uses only bus mastering to move receive data. There is no provision for using slave accesses to move data. When a packet has been completely uploaded, an upComplete indication is normally issued to inform the driver. In some cases, however, the driver would like to be interrupted after *n* number of bytes of a packet have been uploaded (an early interrupt) so that it can examine the packet's header information before continuing with the upload. This

capability is supported through special use of the updNeeded interrupt in the following way:

1. The driver preallocates a one-fragment UPD. The size of the fragment in this UPD should be the desired lookahead size (*n*). If the driver wants to use the RxDiscard command as part of its frame processing, then UPDs should be organized into a ring, even if only one UPD is defined. This implies that no UPD's UpNextPtr entry is ever set to zero.
2. When a packet becomes visible and the upload engine fills the fragment defined in the first UPD, the NIC issues an updNeeded interrupt and goes idle. (These events occur if the frame is larger than the fragment. If the frame completely fits into the lookahead buffer, then an upComplete indication is issued instead of updNeeded.)
3. The driver examines the packet header. It can discard the packet with the RxDiscard command, if desired. If the packet is to be kept, the driver requests buffers from the protocol. When the buffers are returned, a second UPD is filled out with them.
4. The driver resumes the upload by writing the UpListPtr entry with the location of the new UPD and issuing the UpUnStall command. (Or, it can use polling to resume the upload.) Providing the new UPD automatically acknowledges the updNeeded indication.
5. After the upload is complete, the FrameStatus entry is updated as with any other frame. The upPktLength bit contains the received packet length, as usual, but in this case, upPktLength indicates the number of bytes in the frame, not the number of bytes in the second UPD. The difference will be *n* minus the number of bytes placed into the look-ahead UPD.

   If the packet is still being received while it is being uploaded, then the parallel tasking method is being used. If the packet is fully received before the new fragments are made available, then the store-and-forward method is being used. Remember that when reception is finished, the UpPktStatus register contains frame status and size. This information may be of use to the driver, depending on the state of the upload at the time reception is finished.

   From a performance and CPU utilization perspective, this mode probably does not offer any significant advantages over the parallel tasking mode. However, the smaller look-ahead buffer allows a reduced driver size, which may be very important in some environments.

Packets with Errors

Errors are reported in the FrameStatus UPD entry. Possible error conditions are CRC error, aborted frame, implicitly aborted frame, and overrun.

In store-and-forward mode, error information is available before the packet is uploaded. Thus, the driver can simply discard bad frames without uploading them. The NIC itself takes no action on these errors, other than to update RMON counters.

In token ring networks, abnormally large frames can result when stations click in and out of the ring. Through the Command register, the upload engine can be set to limit the amount of data to be uploaded by specifying the maximum number of bytes uploaded for any one packet. When this upload limit has been set, and a packet exceeds this number, the updFull bits are set in all UPDs used by the packet

except the last UPD. In the last UPD, the upComplete and rxOverrun bits are set. Hardware purges the remainder of the packet from the NIC.

NIC Upload Sequence

The NIC performs the following steps to upload a packet to the host:

1. Checks that the UpListPtr register is nonzero.
2. Checks that the NIC is not in the UpStall state.
3. Fetches the FrameStatus entry from the current UPD. If the updComplete or updFull bit is set, the NIC issues an internal UpStall command to stall the upload process. In this case, the driver needs to issue an UpUnStall command before upload can continue. (If the UpPoll register contains a nonzero value, the NIC polls on updComplete and updFull, rather than stalling. When these bits are clear, the upload continues.)
4. Waits for the top receive packet to become visible. This occurs after either 64 bytes have been received or packet reception is completed, whichever comes first.
5. Uploads the packet into the fragments specified in the UPD. If there is more data in the packet than space in the fragment buffers, the NIC sets the updFull bit in the current UPD's FrameStatus entry and continues the upload with the next UPD, if there is one. If there is none, the updNeeded bit is issued and the upload stalls until a new UPD is provided and the UpUnStall command is issued. (As in step 3, polling can be used to unstall the upload.)
6. As the packet is being uploaded, maintains the UpPktStatus register, specifically the upPktLength field. If packet reception is finished, the UpPktStatus register indicates final frame status and received length.
7. At the end of packet upload, updates the UpPktStatus register with any error code from the packet, sets the updComplete bit, and writes UpPktStatus to the UPD's FrameStatus entry.
8. Issues an internal RxDiscard command and waits for it to finish. This does not discard the frame; instead, it enables the NIC to process the next frame in line.
9. If an UpStall command has been carried out, waits until an UpUnStall command has been executed.
10. Fetches the UpNextPtr entry from the UPD. If UpNextPtr is zero and polling is enabled, the NIC starts a polling loop. If polling is disabled, loads the fetched value into the UpListPtr register.
11. Writes UpPktStatus to the UPD in host memory.
12. If a polling loop had been started, polls on UpNextPtr until a nonzero value is fetched, and loads the value into UpListPtr.
13. If the UpListPtr value is zero (polling is disabled), then the upload engine becomes idle and waits for a nonzero value to be written into UpListPtr.
14. Repeats as necessary.

Host Registers

The host registers that apply to upload and reception are described in the following sections.

DmaCtrl

See "DmaCtrl" in Chapter 6.

UpBurstThresh

Synopsis	Provides a threshold that determines when bus master upload requests are made.
Type	Read/write
Size	8 bits
Offset	40

UpBurstThresh Register Format

7	6	5	4	3	2	1	0
0	0	0	0	0			

The UpBurstThresh register determines when the NIC makes upload bus master requests, based on the number of receive frame data bytes in the upload FIFO. The value in UpBurstThresh represents used space in the FIFO in units of 32 bytes. When the used space exceeds the threshold, the NIC may make an upload request on the PCI bus.

For more information about PCI requests, see "PCI Bus Master Operation" in Chapter 3.

A value of zero is invalid. UpBurstThresh defaults to 4, a threshold of 128 bytes.

UpListPtr

Synopsis	Points to the current UPD in the uplist.
Type	Read/write
Size	32 bits
Offset	38

UpListPtr Register Format

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

The UpListPtr register holds the physical address of the current UPD in the uplist. A value of zero in UpListPtr is interpreted by the NIC to mean that no more UPDs are available to accept receive packets.

UpListPtr is cleared by reset, the upDownReset bit, or the upReset bit.

UpListPtr can only point to addresses on 8-byte boundaries, so UPDs must be aligned on 8-byte physical address boundaries.

UpListPtr may be written directly by host software to point the NIC at the head of a newly created uplist. Writes to UpListPtr are ignored by the hardware when the uplist is not empty. When the upload engine is polling on the updFull and updComplete bits, a read of UpListPtr initiates an immediate poll cycle.

UpListPtr is also updated by the NIC as it processes UPDs in the uplist. As the NIC finishes processing a UPD, it loads UpListPtr with the value from the UpNextPtr UPD entry to allow it to move on to the next UPD. If the NIC loads a value of zero

from the current UPD, the upload engine enters the idle state, waiting for a nonzero value to be written to UpListPtr.

To avoid access conflicts between the NIC and host software, the host must issue an UpStall command before writing to UpListPtr.

UpListPtr must be written using a 4-byte command to prevent the upload engine from starting with a transient UpListPtr value.

UpPktStatus

Synopsis	Provides the status of the packet that the upload DMA engine is currently processing (the top packet).
Type	Read-only
Size	32 bits
Offset	30

UpPktStatus Register Format

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

Bits [31:15] are called the *status* field. Bits [14:0] are called the *length* field.

UpPktStatus Bit Descriptions

Bit	Name	Description
[14:0]	upPktLength	These bits, which are called the *length* field, indicate the number of data bytes that have been uploaded at the time this register is read and before the rxComplete bit is set. After rxComplete is set, this field indicates the received size of the packet.
[15]	upStalled	This bit is asserted when the NIC is in the stalled state. There are two ways stall the NIC: ■ Issue an UpStall command. ■ Cause an implicit stall by fetching a UPD when the NIC is not in polling mode and the updComplete or updFull bit in the FrameStatus register is already set. The upStalled bit is cleared with an UpUnStall command.
[18:16]	ARRMatch	These bits indicate the Address Recognition register (ARR) that matched the destination address of the received packet. Only the following codes are used: ■ 000 = ARR0 (this node address) ■ 100 = ARR4 (this node address, for MAC frames) ■ 101 = ARR5 (used for source routing) ■ 111 = ARR7 (used for functional addressing)

(1 of 2)

UpPktStatus Bit Descriptions (continued)

Bit	Name	Description
[19]	rxOverrun	The DMA engine sets this bit to indicate a packet that has overrun. No status other than updComplete is valid if the packet overran. The driver should discard overrunning packets. The rxOverrun bit is set under these conditions: ■ The number of available buffers in NIC memory was insufficient to hold the complete frame. ■ The frame exceeded the limit specified with the SetMaxBytes command.
[21]	rxFc	This is the frame copied bit (fc) of the received FS field. See Appendix A.
[22]	rxAr	This is the address recognized bit (ar) of the received FS field. See Appendix A.
[23]	upPktComplete	This bit is set after upload of the packet is finished.
[24]	updNeeded	This bit is set if uploading of the packet requires a UPD, but either the uplist is empty or the updComplete or updFull bit of the current UPD is set. The setting of this bit is not a fatal condition. When it is set, the upload process stalls, but the remainder of the frame accumulates in NIC memory. The upload can be resumed after the driver provides more UPD fragments and informs the NIC of such. For more information, see "Early Interrupts" earlier in this chapter. This is the same bit that is reported in the IntStatus register, but it is not latched. It is cleared with an RxDiscard command, whereas the bit in the IntStatus register must be acknowledged through the AckInterrupt command.
[26:25]	receiveStatusCode	This field provides encoded receive status as follows: ■ 00 = No error (normal completion) ■ 01 = An aborted packet was received ■ 10 = An implicit abort was received for one of the following reasons: The MAC detected a BURST4 condition. The -REDY signal was lost from the PHY. No EFD was received. ■ 11 = Either the ar bit pair or the fc bit pair in the FS field did not match.
[27]	sourceRouteCompare	When set, this bit indicates that the received packet passed the source route filter.
[28]	redi	(Returned error detect indication.) When set, this bit indicates that a remote station has detected a CRC or code violation in the packet. It is the edi bit of the EFD field (see "SFD and EFD Fields" in Appendix A).
[29]	rlpedi	(Receive local packet error detect indication.) When set, this bit indicates that the receiver detected a CRC or code violation while it was receiving the packet.
[30]	groupAddress	When set, this bit indicates that the received packet has a group address.
[31]	broadcastAddress	When set, this bit indicates that the received packet has a broadcast address.

(2 of 2)

Processing of a packet is not complete until an RxDiscard command has been issued, regardless of whether all the packet data has been uploaded. The hardware issues an RxDiscard command automatically when a packet has been completely uploaded. The driver must issue an RxDiscard command when a packet is to be discarded without being uploaded. The UpPktStatus register is only valid if the packet has been fully received, meaning that the rxComplete bit in the IntStatus register is set. UpPktStatus is written to the last UPD used for the frame at the end of the upload, and then this register is zeroed by the subsequent RxDiscard command.

UpPoll

Synopsis	Sets the polling rate.
Type	Read/write
Size	8 bits
Offset	3D

UpPoll Register Format

7	6	5	4	3	2	1	0
0							

The value in the UpPoll register determines the rate, in 2-microsecond time intervals, at which the current UPD is polled when the NIC is looking for the updComplete bit in the FrameStatus UPD entry to be cleared. The maximum representable value is 254 microseconds.

Polling is disabled when UpPoll is cleared. UpPoll is cleared with a hardware reset, or by setting the upDownReset or upReset bits.

MAC ASIC Registers

The MAC ASIC registers that apply to upload and reception are described in the following sections.

RxBufArea

Synopsis	Specifies the location of the receive buffer area in the host address space.
Type	Write-only
Size	16 bits
Local address	CDE10h

RxBufArea Register Format

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

The host must program the RxBufArea register with 00h before opening the NIC.

RxEarlyThresh

Synopsis	Specifies the number of bytes of a frame that the MAC ASIC must receive before the PCI bridge ASIC is notified of the frame.
Type	Read/write
Size	16 bits
Local address	CDE12h

RxEarlyThresh Register Format

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

At initialization, the driver should program the RxEarlyThresh register for a 64-byte threshold (write 0020h).

8 INTERRUPTS AND INDICATIONS

This chapter provides an overview of interrupts and indications, and defines the registers associated with interrupts.

Indications are reports of any interesting events on the NIC. An indication appears as a set bit in the IntStatus register. Indications can be individually masked off to prevent them from appearing as set in IntStatus. Any indication can be individually configured to cause an interrupt, which is the actual assertion of the interrupt signal on the PCI bus. In this technical reference, the term *interrupt* is used loosely to refer to both interrupts and indications. It is assumed that a driver configures the NIC to generate an interrupt for any indication that is of interest to it.

When responding to an interrupt, the host reads the IntStatus register to determine the cause of the interrupt. The IntStatus register bits define the source of the interrupt. The least-significant bit, interruptLatch, is always set whenever the interrupt pin is asserted. This is done to prevent spurious interrupts on the host bus. The interruptLatch bit must be explicitly acknowledged (cleared) using the AckInterrupt command.

The host acknowledges interrupts by carrying out the interrupt-specific actions summarized in Table 18.

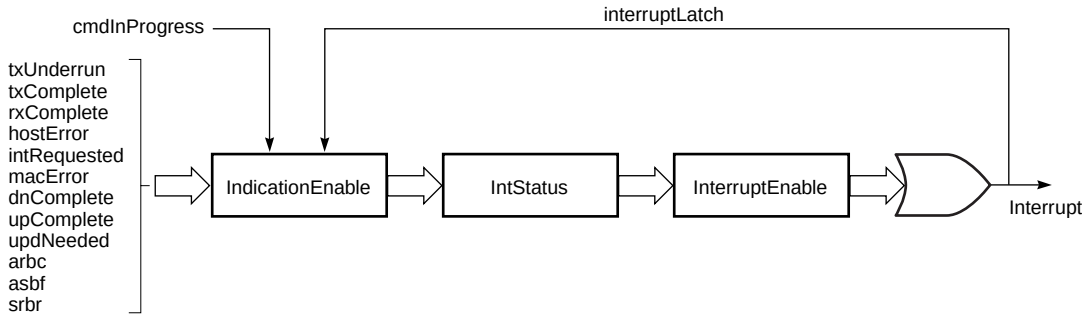
Table 18 Interrupt-specific Actions

Action	Description
interruptLatch	Acknowledged by the AckInterrupt command with the interruptLatchAck bit set
hostError	Acknowledged by issuing the appropriate resets
txComplete	Acknowledged by the AckInterrupt command with the txCompleteAck bit set
txUnderrun	Acknowledged by the DnReset command
rxComplete	Acknowledged by the RxDiscard command
intRequested	Acknowledged by the AckInterrupt command with the intRequestedAck bit set
macError	Acknowledged by the GlobalReset command
dnComplete	Acknowledged by the AckInterrupt command with the dnCompleteAck bit set
upComplete	Acknowledged by the AckInterrupt command with the upCompleteAck bit set
updNeeded	Acknowledged by providing a UPD for the upload or by the RxDiscard command
arbc	Acknowledged by the AckInterrupt command with the arbcAck bit set
asbf	Acknowledged by the AckInterrupt command with the asbfAck bit set
srbr	Acknowledged by the AckInterrupt command with the srbrAck bit set

Interrupt and Indication Enables

An interrupt is an asynchronous indication that an event which requires the attention of the host system has occurred on the NIC. Figure 12 illustrates the relationship between interrupts and indications, and their respective enables. The host uses the IntStatus register to view the various interrupt bits.

Figure 12 Relationship Between Interrupts and Indications



Enables have an immediate effect on indications and interrupts. If a particular interrupt is pending and the host clears its enable bit in the IndicationEnable register, the indication, though still pending, appears as a zero in the IntStatus register and no longer contributes to the assertion of the interrupt line on the host bus (although the interrupt line stays asserted because of the interruptLatch bit).

Conversely, if a pending indication is enabled by setting its enable bit, the indication causes the NIC to assert the interrupt signal on the host bus. The interruptLatch bit is always enabled.

Masking an interrupt or indication does not acknowledge the interrupting event.

The cmdInProgress bit cannot cause an interrupt because its interrupt enable bit is hard-coded to 0. It is merely reported in the IntStatus register.

Most interrupts can be acknowledged through the AckInterrupt command. The interrupt source and the interruptLatch bit can be acknowledged at the same time. Some interrupt sources are acknowledged by means other than AckInterrupt. For example, the txUnderrun interrupt is acknowledged by issuing the DnReset command. DnReset clears the txUnderrun bit in IntStatus, but the interruptLatch bit is not affected; it needs to be cleared with an AckInterrupt command.

Host Registers

The host registers that apply to interrupts and indications are described in the following sections.

IndicationEnable

Synopsis	Specifies which bits in the IntStatus register can become set.
Type	Read-only
Size	16 bits
Offset	5C

IndicationEnable Register Format

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

Each bit set in the IndicationEnable register enables the corresponding bit to be set in the IntStatus register. This register is set using the SetIndicationEnable command. See “SetIndicationEnable” in Chapter 9 for more details.

IndicationEnable is cleared upon hardware reset or with the hostReset bit.

InterruptEnable

Synopsis	Specifies which bits in the IntStatus register can generate an interrupt to the host.
Type	Read-only
Size	16 bits
Offset	5A

InterruptEnable Register Format

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

Each bit in the InterruptEnable register is the interrupt enable bit for the corresponding bit in the IntStatus register. Setting a bit in InterruptEnable allows that source to generate an interrupt on the bus. This register is set using the SetInterruptEnable command. See “SetInterruptEnable” in Chapter 9 for more details.

InterruptEnable is cleared upon hardware reset or with the hostReset bit.

IntStatus

Synopsis	Provides interrupt status and the status of a command that may be in progress.
Type	Read-only
Size	16 bits
Offset	5E

IntStatus Register Format

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

IntStatus Bit Descriptions

Bit	Name	Description
[0]	interruptLatch	This bit is set when the NIC is driving the bus interrupt signal. It is a logical OR of the interrupt-causing bits after they have been filtered through the InterruptEnable register. It is acknowledged by issuing the AckInterrupt command with the interruptLatchAck bit set.
[1]	hostError	This bit is set when a catastrophic error related to the bus interface occurs. The errors that set hostError are PCI target abort; PCI master abort; and false transmit underrun. The GlobalReset command with bits 8, 5, and 3 clear acknowledges the hostError bit.
[2]	txComplete	This bit is set when transmission is complete for a frame in which the txIndicate bit in the FrameStartHeader DPD entry was set. It is cleared by issuing the AckInterrupt command with the txCompleteAck bit set. The PCI bridge ASIC does not know when the transmission is actually complete because of pipelining within the transmitter itself. The ASIC only knows when the last byte was read from the download FIFO. At 4 Mbps, the time between when the MAC ASIC DMA engine last reads data and the transmission of the frame start byte can be as long as 30 microseconds.
[3]	updNeeded	This bit, if enabled, is set whenever the upload needs a UPD and one is not available. A UPD is needed whenever there is a frame to upload but either the uplist is empty or the next UPD in the uplist is not available because it is full. This bit is acknowledged (cleared) when the driver provides a UPD (allowing upload to resume) or an RxDiscard command is issued.
[4]	rxComplete	This bit is set when the NIC has received at least one receive packet completely. It is acknowledged by uploading all of the complete packets, or discarding them without uploading.
[6]	intRequested	This bit is set when the Countdown register has expired or when the InterruptRequest command is issued. It is acknowledged by issuing the AckInterrupt command with the intRequestedAck bit set.
[7]	macError	This bit is set when bit 14, 3, or 2 in the MacStatus register is set. It indicates that an unusual, and perhaps fatal, error has occurred. To restore proper operation, the MAC ASIC needs to be reset and initialized. The macError bit is acknowledged by the GlobalReset command.

(1 of 2)

IntStatus Bit Descriptions (continued)

Bit	Name	Description
[9]	dnComplete	This bit indicates that a packet download has been completed, and that the DPD has had the dnIndicate bit set in its FrameStartHeader entry. This bit can be acknowledged by an AckInterrupt command with the dnCompleteAck bit set. The host should examine the DnListPtr register to determine which packets have been downloaded; those in the downlist before the current DnListPtr (which, if zero, implies all those in the list) have already been downloaded.
[10]	upComplete	This bit indicates that a packet upload has been completed. It is acknowledged by an AckInterrupt command with the upCompleteAck bit set.
[11]	txUnderrun	This bit is set when a transmission has experienced an underrun (the host was not fast enough to keep up with the transmitter). An underrun halts the transmitter and download FIFO, and stalls the download engine. The CP sets this bit, and it should set it only after the transmit path through the MAC ASIC has been restored. It is cleared by the DnReset command.
[12]	cmdInProgress	This bit, when set, indicates that the NIC is still executing the last command issued. It need only be checked after one of the commands that require longer than a single I/O cycle to finish has been issued. No new commands can be issued until cmdInProgress is negated.
[13]	asbf	The CP sets this bit when it has read the response provided in the adapter status block (ASB), and the ASB is available for another response. It is set only if the host has set the asbfr bit in the MISR register, or if an error has been detected in the response. This bit is acknowledged by an AckInterrupt command with the asbfAck bit set.
[14]	srbr	The CP sets this bit when it has recognized a system request block (SRB) request and it has set the return code in the SRB. This bit is acknowledged by an AckInterrupt command with the srbrAck bit set.
[15]	arbc	The CP sets this bit to indicate that the adapter request block (ARB) contains a command for the host to act upon. It is acknowledged by an AckInterrupt command with the arbcAck bit set.

(2 of 2)

IntStatusAuto

Synopsis	Special version of the IntStatus register with some added side effects to allow a reduction in the number of I/O operations required to service interrupts.
Type	Read-only
Size	16 bits
Offset	56

IntStatusAuto Register Format

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

The IntStatusAuto register has the same bit definition as the IntStatus register. It differs from IntStatus only in the side effects that occur when it is read.

In addition to returning the IntStatus value, the following side effects occur when IntStatusAuto is read, preventing further interrupts from occurring:

- The InterruptEnable register is cleared to prevent subsequent events from generating an interrupt on the bus.
- The following bits in IntStatus (if they are set) are acknowledged (cleared): dnComplete, txComplete, upComplete, intRequested, interruptLatch, asbf, srbr, and arbc.

MAC ASIC Registers

The MAC ASIC registers that apply to interrupts and indications are described in the following sections.

MISR

Synopsis	Interrupts the CP.
Type	Read/write/set/clear
Size	8 bits
Local address	CDE0Bh

The MISR register has bit set/clear capability in addition to read/write access. The read/write and set/clear capabilities are encoded in the local bus address. See Table 19.

Table 19 MISR Local Bus Memory Address Bit Definitions

A23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	A0
0	0	0	0	1	1	0	0	1	1	0	1	1	1	1	0	0	command		0	1	0	1	1

MISR Bit Local Bus Memory Address Bit Descriptions

Bit	Name	Description
[6:5]	command	These bits select the MMIO operation to be performed, as follows: ■ 00 = Reads or writes the MISR register using the MacAccessCmd register. A read is performed by issuing a read command. Likewise, a write is performed if a write command is executed. ■ 01 = Performs an AND (or CLEAR) function on the MISR register when a write command is issued. MacData register bits set to 0 during the write cause the corresponding MISR bits to be cleared. MacData bits set to 1 have no effect on the state of the corresponding MISR register bits. Reads with this command are not defined. ■ 10 = Performs an OR (or SET) function on the MISR register when a write command is issued. MacData register bits set to 1 during the write cause the corresponding MISR bits to be set. MacData bits set to 0 have no effect on the state of the corresponding MISR bits. Reads with this command are not defined. ■ 11 = Not defined.

In most cases, the driver should use only the set capability. Bit setting and clearing is accomplished during a memory write access by interpreting a field within the local bus memory address as a command. The value of the data byte being written during the access specifies which bits in the register the command is to be performed upon.

MISR Register Format

7	6	5	4	3	2	1	0

MISR Bit Descriptions

Bit	Name	Description
[5]	csrb	The host is informing that NIC that it has placed a command in the system request block (SRB).
[4]	rasb	The host is informing the NIC that it has placed a response to an adapter request block (ARB) request in the adapter status block (ASB).
[3]	srbfr	The host wants to use the SRB, but the NIC is still processing a previous request. After the SRB return code field has been set, the NIC returns an srbfr interrupt.
[2]	asbfr	The host wants to use the ASB, but the NIC is still processing a previous response. After the ASB return code field has been set, the NIC returns an asbfr interrupt.
[1]	arbf	The host has read the command in the ARB, and the ARB is available.

MacStatus

Synopsis	Provides status for the MacError interrupt reported in the IntStatus register.
Type	Read/write
Size	16 bits
Local address	CDE08h

MacStatus Register Format

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

Although the MacStatus register is read/write, host software must never write it.

Bits 15 and 8 are reserved for PCI bridge ASIC to MAC ASIC communication. Their values are indeterminate at any given time.

MacStatus Bit Descriptions

Bit	Name	Description
[14]	tchk	The firmware writes this bit if an unrecoverable error occurs.
[3]	eint	The hardware sets this bit if the internal watchdog timer has expired. This indicates that the firmware has stopped executing.
[2]	aint	This bit is set if host software performs one of the following illegal operations: ■ Any host write to a write-protected location in private memory, unless the memWrEn bit 6 of the CPAttention register is set. ■ Any host write to the WRBR, WWCR, or WWOR registers, unless the memWrEn bit 6 of the CPAttention register is set.

9 COMMAND REGISTER

This chapter provides an overview of the host Command register and gives definitions of the commands.

Command

Synopsis	Allows commands to be issued to the NIC.
Type	Write-only
Size	16 bits
Offset	5E

Command Register Format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Command Code					Parameter										

The Command register is used to issue commands of various types to the NIC. Most commands execute in less time than it takes for the host system to perform a subsequent read or write operation and are considered to execute in zero time.

In general, a 16-bit access is required when writing the Command register. However, if a particular command has X values occupying the least-significant byte of the command, a byte write to the most-significant byte of the Command register is sufficient. The read-only IntStatus register is located at the same address as the Command register.

The command definitions in this chapter use the following conventions:

- The bit value is the 16-bit value that the NIC expects to be written to the Command register to carry out the desired operation. The Command Code (bits [15:11]) defines the command to be executed. Commands may or may not contain parameters in bits [10:0].
- Bit positions occupied by an "X" indicate that the value for the corresponding bit does not matter. However, for future hardware compatibility it is recommended that zeros be written to these positions.
- Bit positions occupied by a dot (•) indicate bit positions that are to be filled by the parameter associated with the command.

*Commands marked with an asterisk (*) (for example, GlobalReset *) may not be completely executed immediately. For these commands, the driver must ensure that the cmdInProgress bit in the IntStatus register is a zero before taking any further action with the NIC.*

The commands are summarized in Table 20 and described in the following sections.

Table 20 Command Summary

Command Type	Command Name	Bit Value	Description
Reset	GlobalReset	(0000 000• 00•0 ••••) *	Perform an overall reset of the NIC.
	UpReset	(0010 100• 0000 •XXX) *	Reset the upload logic.
	DnReset	(0101 100• 0000 •XXX) *	Reset the download logic.
Transmit	DnStall	(0011 0XXX XXX0 0010)*	Stall the download engine.
	DnUnStall	(0011 0XXX XXX0 0011)	Unstall the download engine.
	SetTxStartThresh	(1001 1••• •••• ••••)	Set the value of the TxStartThresh register.
	DnDisable	(0101 0XXX XXXX XXXX)*	Disable the local download engine.
	DnEnable	(0100 1XXX XXXX XXXX)	Enable the local download engine.
Receive	RxDiscard	(1100 0XXX XXXX XXXX)*	Discard the top receive frame.
	SelectHashFilterBit	(1100 1•XX XX•• ••••)	Program a particular bit in the hash filter.
	UpStall	(0011 0XXX XXX0 0000)*	Stall the upload engine.
	UpUnStall	(0011 0XXX XXX0 0001)	Unstall the upload engine.
Interrupt	AckInterrupt	(0110 1••• X••X •XX•)	Acknowledge active interrupts.
	InterruptRequest	(0110 0XXX XXXX XXXX)*	Cause the NIC to generate an interrupt.
	SetIndicationEnable	(1000 •••• •••• ••••)	Set the value of the IndicationEnable register.
	SetInterruptEnable	(0111 •••• •••• ••••)	Set the value of the InterruptEnable register.
Configuration	SetConfig	(0100 0XXX XXX0 ••••)	Set the value of the Config register.

Reset Commands

GlobalReset *

This command is not always completely executed before the next command can be issued to the NIC. The driver must ensure that the cmdInProgress bit in the IntStatus register is a zero before taking any further action with the NIC.

Bit Value		(0000 000• 00•0 ••0•) *
Bit	**Name**	**Description**
[0]	phyReset	When this bit is clear, the PWRDN signal to the PHY is asserted for 22 PCI clock cycles.
[2]	macReset	When this bit is set, it masks reset to the MAC ASIC. When clear, the MAC ASIC is reset.
[3]	fifoReset	When this bit is set, it masks reset to the FIFO control logic.
[5]	hostReset	When this bit is set, it masks reset to the bus interface logic. If hostReset is not set, the following registers are cleared: IntStatus, InterruptEnable, IndicationEnable, and Countdown.
[8]	upDownReset	When this bit is set, it masks reset to the upload and download logic. If upDownReset is not set, the following upload and download engines and registers are reset: UpMaxBurst, UpLatency, DnListPtr, UpListPtr, DmaCtrl, and UpPktStatus.

The GlobalReset command resets various parts of the NIC, depending on the bit mask passed in the parameter field. Setting bits in the mask causes reset to specific modules to be masked. When the mask is cleared, the entire NIC is reset, which is equivalent to a hardware reset.

You can also use the MacAccessCmd register to activate resets through the cpHold bit in the Pmbar register.

DnReset *

This command is not always completely executed before the next command can be issued to the NIC. The driver must ensure that the cmdInProgress bit in the IntStatus register is a zero before taking any further action with the NIC.

Bit Value		**(0101 100• 0000 •XXX) ***
Bit	**Name**	**Description**
[3]	dnFifoReset	When this bit is set, it masks reset to the transmit FIFO control logic. If this bit is not set, the transmit FIFO is flushed, and the TxStartThresh register is forced to its reset state.
[8]	dnReset	When this bit is set, it masks reset to the download logic. When it is clear, the download logic is reset (the DnListPtr and DnPoll registers and the dnComplete and dnInProg bits in the DmaCtrl register are reset).

Relative to download, this command is identical to the GlobalReset command with the upDownReset bit clear.

UpReset *

This command is not always completely executed before the next command can be issued to the NIC. The driver must ensure that the cmdInProgress bit in the IntStatus register is a zero before taking any further action with the NIC.

Bit Value		**(0010 100• 0000 •XXX) ***
Bit	**Name**	**Description**
[3]	upFifoReset	When this bit is set, it masks reset to the receive FIFO control logic. If this bit is not set, the receive FIFO contents are flushed.
[8]	upReset	When this bit is set, it masks reset to the upload logic. When it is clear, the upload logic is reset (the UpPoll, UpListPtr, and UpPktStatus registers and the upComplete bit in the DmaCtrl register are reset). The upload stall condition is also cleared.

Transmit Commands

DnDisable *

This command is not always completely executed before the next command can be issued to the NIC. The driver must ensure that the cmdInProgress bit in the IntStatus register is a zero before taking any further action with the NIC.

Bit Value	(0101 0XXX XXXX XXXX) *

The DnDisable command disables the local download engine. This command has no effect on the host download, which is controlled by the downlist and the stall condition. This command takes effect only after packet transmission (if any) is complete when the command is issued. This command does not control the transmitter, which is controlled by the Open.Nic command (described in Chapter 11). DnDisable controls only the download function.

DnEnable

Bit Value	(0100 1XXX XXXX XXXX)

The DnEnable command enables the local download engine to queue frames to the MAC ASIC. This does not, by itself, enable the NIC to transmit packets. The NIC must also be open for transmission to occur.

DnStall *

This command is not always completely executed before the next command can be issued to the NIC. The driver must ensure that the cmdInProgress bit in the IntStatus register is a zero before taking any further action with the NIC.

Bit Value	(0011 0XXX XXX0 0010) *

The DNStall command stops the NIC from fetching the DnNextPtr DPD entry and loading it into the DnListPtr register.

If DnListPtr is nonzero, the driver must issue a DnStall command before modifying the downlist to avoid conflicts with the DnListPtr updates. The host must wait for the cmdInProgress bit to be deasserted before continuing.

DnUnstall

Bit Value	(0011 0XXX XXX0 0011)

The opposite of DnStall, this command releases the NIC to fetch the DnNextPtr DPD entry and update the DnListPtr register. The host should issue this command as soon as possible after the DnStall command, once it has finished modifying the downlist.

SetTxStartThresh

Bit Value	(1001 1••• •••• ••••)

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

The NIC queues packets to the transmitter as soon as the number of bytes downloaded to the transmit FIFO is greater than the value in TxStartThresh. If the packet being transmitted is shorter than TxStartThresh, then queueing of the packet commences as soon as the entire packet has been downloaded.

Receive Commands

RxDiscard *

This command is not always completely executed before the next command can be issued to the NIC. The driver must ensure that the cmdInProgress bit in the IntStatus register is a zero before taking any further action with the NIC.

Bit Value	(1100 0XXX XXXX XXXX) *

The RxDiscard command causes the top receive frame to be discarded.

RxDiscard should not be used if the driver organizes UPDs into a grounded chain. They must be organized as a ring, even if only one UPD is defined.

SelectHashFilterBit

Bit Value	(1100 1•XX XX•• ••••)

The SelectHashFilterBit command is used to set individual bits in the hash filter table for multicast packet reception. Each bit in the hash filter corresponds to a set of multicast addresses that can be received. Bits [5:0] select one of the 64 possible entries in the table. Bit 10 specifies whether the selected bit should be cleared or set.

The hash filter acts as an array of 64 enable bits. Incoming frames have a cyclic redundancy check (CRC) applied to their destination address. The low-order six bits of the CRC are used as an index into the hash filter. If the hash filter bit addressed by the index is set, the NIC accepts the packet. If the hash filter bit is cleared, the NIC discards the packet.

In addition, the NIC must be directed to enable multicast reception. This is accomplished through the system request block (see Chapter 11).

UpStall *

This command is not always completely executed before the next command can be issued to the NIC. The driver must ensure that the cmdInProgress bit in the IntStatus register is a zero before taking any further action with the NIC.

Bit Value	(0011 0XXX XXX0 0000) *

The UpStall command stops the NIC from fetching the UpNextPtr UPD entry and loading it into the UpListPtr register. Whenever the host wishes to modify the uplist, and UpListPtr is nonzero, the host must issue an UpStall command to avoid conflicts with the NIC's UpListPtr updates. Note that this command requires the host to wait for cmdInProgress to be deasserted before continuing.

UpUnStall

Bit Value	(0011 0XXX XXX0 0001)

The opposite of UpStall, this command releases the NIC to fetch the UpNextPtr UPD entry and load the UpListPtr register. The host should issue this command as soon as possible after UpStall, once it has finished modifying the uplist.

When the upload engine stalls because it is reading a UPD that is in use (meaning that either the updComplete or updFull bit is set in the UPD FrameStatus entry), the NIC can automatically execute an UpUnStall command by polling on these bits and waiting for the software to clear them. This function is enabled when the UpPoll register contains a nonzero value.

Interrupt Commands

AckInterrupt

Bit Value		(0110 1••• X••X •XX•)
Bit	**Name**	**Acknowledged IntStatus Bit**
[0]	interruptLatchAck	interruptLatch
[1]	txCompleteAck	txComplete
[2]	intRequestedAck	intRequested
[3]	dnCompleteAck	dnComplete
[4]	upCompleteAck	upComplete
[5]	asbfAck	asbf
[6]	srbrAck	srbr
[7]	arbcAck	arbc

The AckInterrupt command resets selected interrupt indications in the IntStatus register. When it is issued, the indications that correspond to bits set to 1 in the parameter field are cleared.

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

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

InterruptRequest *

This command is not always completely executed before the next command can be issued to the NIC. The driver must ensure that the cmdInProgress bit in the IntStatus register is a zero before taking any further action with the NIC.

Bit Value	(0110 0XXX XXXX XXXX) *

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

The 3C359 NIC can generate an automatic intRequested interrupt when the Countdown register count reaches zero. The driver must maintain internal state to determine what to do when an intRequested interrupt occurs.

SetIndicationEnable

Bit Value	(1000 •••• •••• ••••)

The SetIndicationEnable command is used to set or clear bits of the IndicationEnable register. Each bit in the parameter field specifies whether the corresponding bit in the IndicationEnable register is to be set (1) or cleared (0).

Although the bits in the SetIndicationEnable command do not correspond bit-for-bit with those in the IndicationEnable register, the order of bits matches. For example, bit 0 in the SetIndicationEnable command controls the least-significant controllable bit of IndicationEnable (bit 1, hostError) and bit 11 controls the most-significant bit of IndicationEnable (bit 15, arbc). Refer to the IntStatus register definition for the map of the indication bits.

Indications disabled with the SetIndicationEnable command do not cause the indication to appear in the IntStatus register. All indication enables are cleared upon NIC reset. Bits 0 and 12 of the IndicationEnable register cannot be written because their status is always available in IntStatus. Bits 5 and 8 of IndicationEnable have no function.

SetInterruptEnable

Bit Value	(0111 •••• •••• ••••)

The SetInterruptEnable command is used to set or clear bits of the InterruptEnable register. Each bit in the parameter field specifies whether the corresponding bit in InterruptEnable is to be set (1) or cleared (0).

Although the bits in SetInterruptEnable do not correspond bit-for-bit with those in the InterruptEnable register, the order of bits matches. For example, bit 0 in the SetInterruptEnable command controls the least-significant controllable bit of the InterrruptEnable register (bit 1, hostError) and bit 11 controls the most-significant bit of InterrruptEnable (bit 15, arbc). Refer to the IntStatus register definition for the map of the interrupt bits.

Interrupts disabled with the SetInterruptEnable command block the corresponding interrupt from causing an interrupt to the host. All interrupt enables are cleared upon NIC reset. Bits 0 and 12 of the InterruptEnable register cannot be written because the interruptLatch bit in the IntStatus register is always enabled, and the cmdInProgress bit cannot cause an interrupt signal. Bits 5 and 8 of InterruptEnable have no function.

SetConfig

Bit Value	(0100 0XXX XXX0 ••••)

The SetConfig command is used to set bits [3:0] of the Config register. The parameter field specifies the value to be written to Config, as follows:

Bit	Value in the Config Register
[0]	Global enable of the hash filter.
[1]	Configures the maximum number of uploaded bytes to be 8192.
[2]	Configures the maximum number of uploaded bytes to be 20480.
[3]	Configures the download mode. When clear, download restricts itself to one frame at a time in the download FIFO. When set, download strives to keep the FIFO full at all times, regardless of how many frames may be in the FIFO.

10 OTHER REGISTERS

This chapter describes various other registers in the 3C359 NIC.

Config

Synopsis	Contains mode bits that are used to configure the operation of the PCI bridge ASIC.
Type	Read-only
Size	8 bits
Offset	29

Config Register Format

7	6	5	4	3	2	1	0
0	0	0					

Config Bit Descriptions

Bit	Name	Description
[0]	hashEn	When this bit is clear, it indicates that hashing is disabled; when set, hashing is enabled. Proper use of hashing also requires that the hash table be set correctly (see "SelectHashFilterBit" in Chapter 9) and that the MAC has been configured for multicast reception through the software interface system request block (SRB). See Chapter 11 for details on the SRB.
[1]	maxFrameEq8192	When this bit is set, it indicates that the upload engine has been configured to limit the largest packet uploaded to 8,192 bytes. The hardware truncates packets larger than this, and the rxOverrun bit in the FrameStatus field of the last UPD used for the packet (the one with the updComplete bit set) is set.
[2]	maxFrameEq20480	When this bit is set, it indicates that the upload engine has been configured to limit the largest packet uploaded to 20,480 bytes. The hardware truncates packets larger than this, and the rxOverrun bit in the FrameStatus field of the last UPD used for the packet (the one with the updComplete bit set) is set.
[3]	downloadMode	When this bit is clear, the download restricts itself to one packet at a time in the download FIFO. When this bit is set, the download is not restricted to one packet; it attempts to keep the FIFO full regardless of the number of packets.

The SetConfig command is used to set bits in this register.

Countdown

Synopsis	Provides a mechanism for the host to cause the NIC to generate an interrupt in a programmable time period.
Type	Read/write
Size	16 bits
Offset	36

Countdown Register Format

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

The Countdown register is a programmable down-counter that can cause the NIC to generate an interrupt when the counter expires.

The host software loads Countdown with an initial countdown value. Thereafter, Countdown decrements at a rate determined by the counterSpeed bit in the DmaCtrl register. When counterSpeed is clear, the count rate is once every 8 microseconds. When counterSpeed is set, the count rate is once every 2 microseconds. When Countdown reaches zero, it continues to count, wrapping to FFFFh.

Countdown can cause an intRequested interrupt when it counts through zero. The interrupt is generated if the armCountdown bit in the DmaCtrl register is set at the time of the 1-to-0 transition.

The armCountdown bit is managed solely by the hardware according to the following rules:

- Set when a nonzero value is written to Countdown
- Cleared when the value zero is written to Countdown, or when Countdown counts through zero

This means that when the host writes a nonzero value to Countdown, an interrupt is generated in a corresponding amount of time. By writing a zero value to Countdown, the host can suppress interrupts.

Countdown is cleared by hardware reset (hostReset bit in the GlobalReset command).

FreeTimer

Synopsis	Provides a free-running counter to be used for general timing purposes.
Type	Read-only
Size	16 bits
Offset	34

FreeTimer Register Format

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

The FreeTimer register is a free-running, read-only counter that increments at precise time intervals so that it can used for timing measurements. The count interval for FreeTimer is determined by the counterSpeed bit in the DmaCtrl register.

When counterSpeed is cleared, the count rate is once every 8 microseconds (four byte times at 4 Mbps). This yields a maximum measurable time interval of 524 milliseconds. When counterSpeed is set, the count rate is once every 2 microseconds (four byte times at 16 Mbps), giving a maximum measurable time interval of131 milliseconds.

FreeTimer is cleared by hardware reset or the hostReset bit in the GlobalReset command register.

HashFilter

Synopsis	Defines the values for a 64-bit multicast address hash filter.
Type	Read-only
Size	8 bits
Offset	28

HashFilter Register Format

7	6	5	4	3	2	1	0

The value of the hash filter can be read by doing eight successive reads. The low-order eight bits are returned first, followed by the successive higher-order eight bits.

Individual bits in the hash filter are set or cleared using the SelectHashFilterBit command.

HashFilter is cleared by hardware reset.

SwitchSettings

Synopsis	Defines the NIC ring speed.
Type	Read/write
Size	16 bits
Local address	1C88h

SwitchSettings Register Format

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

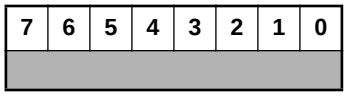
The only bit of interest to the driver is bit 1, ringSpeed. The ringSpeed bit settings are:

- 0 = 16 Mbps
- 1 = 4 Mbps

Timer

Synopsis	Provides a general purpose timer function.
Type	Read-only
Size	8 bits
Offset	1A

Timer Register Format



The Timer register contains an 8-bit counter that begins counting from zero upon the assertion of the interrupt signal. The host can use this function to make interrupt latency measurements. The counter increments by one every 2 microseconds. When the counter reaches FFh, it halts. This yields a maximum measurable interrupt latency of 512 microseconds.

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

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

11 SOFTWARE OPERATION

This chapter describes the software interface, which allows you to perform high-level operations such as inserting the NIC into the ring (opening the NIC) or requesting statistics from the MAC ASIC.

MAC Packets

Although MAC packets are not normally uploaded to the host, the Open.NIC and Modify.Open.Parms commands allow some or all types of MAC frames to be forwarded to the host.

MAC frames are read from the receive buffer in shared memory and placed into buffers in host memory. Bus mastering is not used. The adapter request block (ARB) is used to notify the host that a MAC frame is to be forwarded.

Multicast Reception

The 3C359 NIC can be configured to receive up to 64 different multicasts. These steps enable multicast reception:

1. For each multicast to be received, the driver must calculate the hash for the destination address and set the corresponding bit in the hash table, as described in "Multicast Filtering" in Chapter 7.
2. The driver uses the SetConfig command to set the hashEn bit in the Config register.
3. The driver issues the Set.Multicast.Mode command to the communications processor (CP).
4. The CP configures the MAC ASIC for multicasting and returns a response in the system request block (SRB).

To delete a multicast that was set previously, use the SelectHashFilterBit command with bit 10 cleared and bits [5:0] pointing to the hash table entry to be cleared.

Upon hardware reset, the hash filter is cleared. The driver must reprogram the hash filter, if necessary.

Because more than one destination address may hash to the same 6-bit value, the driver must also filter the address before it decides to accept a frame.

If only one multicast address is to be set, the Set.Group.Address command can be used, which provides better performance than the Set.Multicast.Mode command. When this method is used, it is not necessary to filter the address after reception.

Communication with the Host

Communication between the CP and the host is by means of *control blocks* and buffers in the shared RAM. Commands and their status pass between the CP and the host in control blocks, which, in conjunction with interrupts, provide

event-driven, asynchronous NIC operation. The commands include high-level requests from the host for MAC and LLC services. Use of these requests can greatly reduce host program size and complexity.

The control blocks are summarized in Table 21.

Table 21 Control Blocks

Block	Abbreviation	Use
System request block	SRB	Passes transmit information or commands from the host to the CP.
Adapter request block	ARB	Passes receive information or commands from the CP to the host.
Adapter status block	ASB	Passes host responses to ARB commands to the CP.

SRB Commands

The system request block (SRB) is used to pass a command from the host to the CP. It is located at local address DFE90h. The SRB address is returned in the SRB_ADDRESS field of the open completion response SRB (see Table 30). The NIC supports the SRB commands listed in Table 22.

Table 22 SRB Command Summary

Command	Code (Hex)	Description
Request.Interrupt	00	Causes the NIC to issue an interrupt to the host.
Modify.Open.Parms	01	Allows the host to change operational parameters.
Restore.Open.Parms	02	Modifies the OPEN_OPTIONS parameter set by the Open.NIC command.
Open.NIC	03	Inserts the NIC onto the ring with specified parameters.
Close.NIC	04	Removes the NIC from the ring.
Set.Sleep.Mode	05	Directs the NIC to go into the remote wake-up mode.
Set.Group.Address	06	Sets group addresses.
Set.Funct.Address	07	Sets functional addresses.
Read.Log	08	Resets the statistics counters to zero after reading.
Set.Multicast.Mode	0C	Tells the NIC to enable or disable the multicast function.
Change.Wakeup.Pattern	0D	Notifies the NIC that the host wants to add or delete a wake-up packet pattern.
Get.Statistics	13	Requests a dump of the RMON statistics.
Set.Receive.Mode	1F	Tells the NIC to set different receive modes.

Issuing SRB Commands

A driver uses the following sequence to issue SRB commands to the CP:

1 Set up the MacAccessCmd register as follows:

- opCode = PrivateMemWrite
- localAddress = DFE90h

2 Write the SRB contents to the MacData register. The write operation can be in byte or word increments. This write causes the value in the MacData register to be written to the address specified in the MacAccessCmd register.

3 The NIC does not automatically increment the localAddress after each write access. Therefore, to write the next byte or word, the driver must write the MacAccessCmd register so that the localAddress bits point to the next location to be accessed.

4 After writing the entire SRB contents, the driver interrupts the NIC by setting the csrb bit of the MISR register.

The SRB commands are described in the following sections.

Change.Wakeup.Pattern

The Change.Wakeup.Pattern command notifies the NIC that the host wants to add or delete a wake-up packet pattern. Together with this command, the sleeping system can be awakened by receiving and matching the frame with the specific pattern. See Table 23.

Table 23 Change.Wakeup.Pattern Command Parameters

Offset	Parameter Name	Byte Length	Description
0	COMMAND	1	Hex code = 0D, Change.Wakeup.Pattern.
1		1	Reserved.
2	RETCODE (hex)	1	Set by the NIC upon return. Valid return codes are: ■ 00 = Operation completed successfully ■ 01 = Invalid command code ■ 06 = Invalid options ■ 46 = Not available
3		1	Reserved.
4	PATTERN_FLAG (hex)	2	Add or delete pattern: ■ 0001 = Add ■ 0002 = Delete
6	MASKSIZE	2	The size, in bytes, of the pattern mask. The range is 0 to 12 bytes.
8	PATTERNOFFSET	2	The offset from the beginning of the scratch buffer to the packet pattern (0 to 255 bytes).
10	PATTERNSIZE	2	The size, in bytes, of the packet pattern. The range is 1 to 96 bytes.
12	MASK	12	The bit mask, which can be up to 96 bits wide. The bits are set as follows: ■ Bit *n* = 1, if host wants to ignore the *n*th byte comparison in the packet pattern. ■ Bit *n* = 0, if the host wants to compare the *n*th byte in the packet pattern.

The host can issue multiple Change.Wakeup.Pattern commands to add or delete multiple packet patterns.

The Change.Wakeup.Pattern command can be issued before the NIC is opened. Before issuing this command, the driver must write the pattern of the frame to the NIC's scratch buffer, which is located at local address DFEF0h of the NIC SRAM. The NIC reserves a buffer space large enough to hold a frame pattern at this location.

This command is needed if the host wants to use the packet pattern-matching capability to wake up the sleeping system.

Close.NIC The Close.NIC command closes the NIC and terminates all ring communication. This command can be issued any time after the NIC has been initialized. Commands the NIC has accepted are not completed, and they are not returned to the host. The NIC is removed from the ring, and the WRBR register is reset to the value it had before the Open.NIC command was issued. When the NIC completes the operation, it sets the RETCODE and interrupts the host. See Table 24.

Table 24 Close.NIC Command Parameters

Offset	Parameter Name	Byte Length	Description
0	COMMAND	1	Hex code = 04, Close.NIC.
1		1	Reserved.
2	RETCODE (hex)	1	Set by the NIC upon return. Valid return codes are: ■ 00 = Operation completed successfully ■ 01 = Invalid command code

Get.Statistics The Get.Statistics command obtains MIB management information statistics counters that the NIC maintains. The NIC must be opened before this command is issued. The NIC copies the MIB statistics counters into the scratch buffer located at DFEF0h of the SRAM. The NIC then clears the counters before returning an SRB completion indication to the driver. When the driver receives the SRB completion indication, it can use the MacAccessCmd and MacData registers to read the counters from the scratch buffer. See Table 25.

Table 25 Get.Statistics Command Parameters

Offset	Parameter Name	Byte Length	Description
0	COMMAND	1	Hex code = 13, Get.Statistics.
1		1	Reserved.
2	RETCODE (hex)	1	Set by the NIC upon return. Valid return codes are: ■ 00 = Operation completed successfully ■ 01 = Invalid command code ■ 04 = NIC closed; it should be open

The NIC maintains the MIB statistics counters and copies them to the scratch buffer in the order shown in Table 26.

Table 26 MIB Statistics Counters

Offset	Counter Name	Byte Length	Remarks
0	Total Bytes Received	4	LLC frames only.
4	Total Frames Received	4	LLC frames only.
8	Total Bytes Transmitted	4	LLC frames only.
12	Total Frames Transmitted	4	LLC frames only.
16	Receive CRC Errors	2	Received LLC frames only.
18	MAC Frames Received	2	
20	Function MAC Received	2	
22	Group MAC Received	2	
24	MAC Frames Transmitted	2	
26	Group MAC Transmitted	2	
28	PTT Time-out Errors	2	MAC and LLC frames.
30	Transmit Underrun Errors	2	MAC and LLC frames.
32	Signal Loss Errors	2	

After the Get.Statistics command is issued, the CP resets the associated counters to zero. The driver must keep track of the values it reads.

Other statistics counters are available through the Read.Log command.

Modify.Open.Parms

The Modify.Open.Parms command modifies the Open.NIC command OPEN_OPTIONS parameter. When the NIC completes the operation, it sets the RETCODE and interrupts the host. See Table 27.

Table 27 Modify.Open.Parms Command Parameters

Offset	Parameter Name	Byte Length	Description
0	COMMAND	1	Hex code = 01, Modify.Open.Parms.
1		1	Reserved.
2	RETCODE (hex)	1	Set by the NIC upon return. Valid return codes are: ■ 00 = Operation completed successfully ■ 01 = Invalid command code ■ 04 = NIC closed; it should be open
3		1	Reserved.
4	OPEN_OPTIONS	2	See Table 29.

Open.NIC

The Open.NIC command inserts the NIC onto the ring with specified parameters. See Table 28.

Table 28 Open.NIC Command Parameters

Offset	Parameter Name	Byte Length	Description
0	COMMAND	1	Hex code = 03, Open.NIC.
1		7	Reserved.
8	OPEN_OPTIONS	2	See Table 29.
10	NODE_ADDRESS	6	This NIC's ring address. If 0, the address is the node address. The byte order of the ring address must be swapped from the IEEE byte order. For example: Bytes: 10 12 14 IEEE address: 6655 4433 2211 Ring address: 5566 3344 1122
16	GROUP_ADDRESS	4	The group address to set. If 0, no address is set. The byte order of the group address must be swapped from the IEEE byte order. For example: Bytes: 16 18 IEEE address: 6655 4433 Ring address: 5566 3344 See also "Set.Group.Address" later in this chapter.
20	FUNCT_ADDRESS	4	The function address to set. If set to 0, no address is set. The byte order of the function address must be swapped from the IEEE byte order. For example: Bytes: 20 22 IEEE address: 6655 4433 Ring address: 5566 3344 See also "Set.Funct.Address" later in this chapter.
24		8	Reserved.
32	MAC_PROTOCOL	1	Allows the driver to specify a specific MAC access protocol for the NIC, as follows: ■ 0 = TXI/TKP protocol. The NIC tries to access the ring using the TXI protocol. If the access fails, then the NIC uses the TKP protocol. TXI is the default. ■ 1 = TKP only. The NIC uses the classic TKP protocol. ■ 2 = TXI only. The NIC uses the TXI protocol.

Table 28 Open.NIC Command Parameters (continued)

Offset	Parameter Name	Byte Length	Description
33	SUPPRESS_ERROR	1	Allows the driver to specify whether the NIC should generate a receiver congestion error MAC frame when a receive overrun condition occurs, as follows: ■ 0 = The NIC generates a receiver congestion error MAC frame when a receive overrun occurs (default). ■ 1 = The NIC does not generate a receiver congestion error MAC frame.
34	N/A	8	Reserved.
42	N/A	18	Reserved.

(2 of 2)

Table 29 OPEN_OPTIONS Bit Descriptions

Bit	Name	Description
[0]	contender	When this bit is set, the NIC participates in monitor contention (claim token) when the opportunity arises. When this bit is off, the NIC does not participate. If the NIC detects the need for a new active monitor, it initiates claim token processing regardless of whether this bit is set on or off.
[1]	Reserved	Bit value must be 0.
[2]	Reserved	Bit value must be 0.
[3]	passAttentionMacFrames	When this bit is set, the NIC passes directly to the host all attention MAC frames that are not the same as the last attention MAC frame received. When this bit is off, these frames are not passed.
[4]	passNicMacFrames	When this bit is set, the NIC passes directly to the host all NIC class MAC frames that are received but not supported by the NIC. When this bit is off, these frames are ignored.
[5]	disableSoftError	When this bit is set, it prevents soft error status changes from causing interrupts.
[6]	disableHardError	When this bit is set, it prevents hard error and transmit beacon status changes from causing interrupts.
[7]	wrapInterface	When this bit is set, the NIC does not attach itself to the network. Instead, it wraps all transmitted data as received data.
[10:8]	Reserved	Bit values must be 0.
[11]	duplex	This bit allows the driver to specify full-duplex or half-duplex frames as follows: ■ 0 = Full-duplex mode (default) ■ 1 = Half-duplex mode
[12]	tokenRelease	This bit is only available when the NIC is operating at 16 Mbps. When it is set to 0, the NIC gets early token release as the default. When set to 1, the NIC gets no early token release as the default.

(1 of 2)

Table 29 OPEN_OPTIONS Bit Descriptions (continued)

Bit	Name	Description
[13]	remoteProgramLoad	This bit prevents the NIC from becoming a monitor during the open process. If it is set to 1, the NIC fails the open process if there is no other NIC on the ring when it attempts to insert into the ring.
[14]	Reserved.	Bit value must be 0.
[15]	passBeaconMacFrames	When this bit is set, it passes directly to the host the first beacon MAC frame and all subsequent beacon MAC frames that have a change in the source address or the beacon type.

(2 of 2)

When the NIC completes the Open.NIC command, it generates a system request block (SRB) response with the format shown in Table 30.

Table 30 SRB Response Format

Offset	Parameter Name	Byte Length	Description
0	COMMAND	1	Hex code = 03, Open.NIC.
1		1	Reserved.
2	RETCODE (hex)	1	Code set by the NIC upon return.
3		4	Reserved.
7	OPEN_ERROR_CODE	1	Valid if RETCODE is 07h. See "Open Errors" later in this chapter.
8	ASB_ADDRESS	2	Address of the beginning of the ASB.
10	SRB_ADDRESS	2	Address of the beginning of the SRB.
12	ARB_ADDRESS	2	Address of the beginning of the ARB.
14	VERSION_STRING	2	Contains an offset to an ASCII character string in the NIC SRAM that tells the revision number and the date of the microcode that the NIC is running. The version string has the following format: rr.rr mm/dd/yy where rr.rr is the revision number in decimal and mm/dd/yy is the date of the microcode in month/date/year format. The driver can use the MacAccessCmd and MacData registers to read the version string.
16	DTR_FLAG	1	Indicates the MAC protocol that the NIC is using, as follows: ■ 0 = The NIC has opened using the TKP protocol. ■ Nonzero = The NIC is attached to a DTR switch and opened using the TXI protocol.

Rules for TXI Protocol

When the NIC is using the TXI access protocol, the driver must loop back LLC frames whose destination addresses meet any of the following conditions:

- Matches the source address
- Is a broadcast address
- Matches the station's function address
- Matches the station's group address

Open Errors

Open errors are returned in two bytes. The high-order byte is always 0. The low-order byte contains the following:

- In the high-order half: the test phase in which the error was encountered.
- In the low-order half: the error condition.

Table 31 summarizes the open error code values. Table 32 summarizes responses to open error codes. For more information on open errors, see *IBM Local Area Network Technical Reference, Appendix B, Return Codes (November 1988)*.

Table 31 Open Error Code Values

Value	Test Phase	Value	Error
1n	Lobe media test	n1	Function failure
2n	Physical insertion	n2	Signal loss
3n	Address verification	n3	Reserved
4n	Roll call poll (neighbor notification)	n4	Reserved
5n	Request parameters	n5	Timeout
		n6	Ring failure
		n7	Ring beaconing
		n8	Duplicate node address
		n9	Parameter request
		nA	Remove received
		nB	Reserved
		nC	Reserved
		nD	No monitor detected
		nE	Monitor contention failed for RPL

Table 32 Responses to Open Error Codes

Code (Hex)	Description	Action (See Number in Table 33)
11	Lobe media function failure. The lobe bit-error rate is too high, or the NIC is unable to receive.	1, 3, 5
26	Physical insertion ring failure. The active monitor NIC was unable to complete ring purge.	1, 2a

Table 32 Responses to Open Error Codes (continued)

Code (Hex)	Description	Action (See Number in Table 33)
27	Physical insertion, ring beaconing. The NIC tried to open on a ring operating at a different data rate, a monitor contention (claim token) failure occurred, or the NIC received a beacon MAC frame from the ring.	1, 2, 2b
2A	Physical insertion timeout. A network management function directed the NIC to get off the ring.	2a, 4
2D	No monitor detected. The RPL station is the first attempting to insert onto the ring.	1, 2a
2E	Monitor contention failed for RPL.	2
32	Address verification signal loss. A 250-millisecond signal loss occurred after the NIC completed ring signal recognition.	1, 2a
35	Address verification timeout. The insertion timer expired before this function was completed. The ring may be congested, experiencing a high bit-error rate, or losing enough tokens or frames to prevent successful transmission of address verification MAC frames.	1, 2a
36	Address verification ring failure. Acting as an active monitor, the NIC was unable to complete the ring purge function. An error condition occurred after the successful completion of a monitor contention (claim token), when the NIC became an active monitor.	1, 2a
37	Address verification ring beaconing. The NIC detected a monitor contention (claim token) failure or received a beacon MAC frame from the ring.	1, 2b
38	Address verification, duplicate node address. The NIC detected that another station on the ring has a NIC with the same address.	4
3A	Address verification, remove received. A network management function directed the NIC to get off the ring.	2a, 4
42	Ring poll signal loss. A 250-millisecond signal loss occurred after the NIC completed ring signal recognition.	1, 2a
45	Ring poll timeout. The insertion timer expired before this function was completed. The ring may be congested, experiencing a high bit-error rate, or losing enough tokens or frames to prevent successful reception of the ring poll request or MAC frame, or transmission of the required ring poll response MAC frame.	1, 2a
46	Ring poll failure. Acting as an active monitor, the NIC was unable to complete the ring purge function. An error condition occurred after the successful completion of a monitor contention (claim token), when the NIC became an active monitor.	1, 2a
47	Ring poll ring beaconing. The NIC detected a monitor contention (claim token) failure or received a beacon MAC frame from the ring.	1, 2b
4A	Ring poll remove received. A network management function directed the NIC to get off the ring.	2a, 4
55	Request parameters timeout. The insertion timer expired before this function was completed. The ring may be congested, experiencing a high bit-error rate, or losing enough tokens or frames to prevent successful transmission of the request parameter MAC frame or reception of either the set parameters 1 or set parameters 2 MAC frame (required response to the NIC request).	1, 2a

Table 32 Responses to Open Error Codes (continued)

Code (Hex)	Description	Action (See Number in Table 33)
56	Request parameters ring failure. Acting as an active monitor, the NIC was unable to complete the ring purge function. An error condition occurred after the successful completion of a monitor contention (claim token), when the NIC became an active monitor.	1, 2a
57	Request parameters ring beaconing. The NIC received a beacon MAC frame from the ring.	1, 2b
59	Request parameters, parameter request. The NIC detected that the ring parameter server is present on the ring, but that the required response (set parameter 1 or set parameter 2 MAC frame) was not received in time. The ring may be congested, experiencing a high bit-error rate, or losing too many tokens or frames.	1, 2a
5A	Request parameters, remove received. A network management function directed the NIC to get off the ring.	2a, 4

(3 of 3)

Table 33 Open Error Actions

Number	Description
1	Delay 30 seconds or more, retry the open two times, with a 30-second delay between the retries.
2	Delay 30 seconds or more, check NIC configuration for data rate, and retry the open.
2a	If error persists, direct the PC system operator to contact the network system administrator for assistance and provide open error information.
2b	If error persists, direct the PC system operator to contact the network system administrator for assistance and provide NIC status parameter information.
3	Direct the PC system operator to contact the network system administrator for assistance and provide open error information.
4	Direct the PC system operator to contact the network system administrator for assistance, provide node address information, and try attaching to the ring again after 6 minutes.
5	If this error persists, there is a problem with the NIC or the lobe.

Read.Log

The Read.Log command reads and resets NIC error counters. The Read.Log command parameters are listed in Table 34. This command should be issued if the NIC receives a ring status ARB with the error counter overflow set, which occurs if a NIC error counter reaches a count of 255. The error counters are described in Table 35. The Read.Log command can be issued any time between the Open.NIC and Close.NIC commands.

Table 34 Read.Log Command Parameters

Offset	Parameter Name	Byte Length	Description
0	COMMAND	1	Hex code = 08, Read.Log.
1		1	Reserved.

(1 of 2)

Table 34 Read.Log Command Parameters (continued)

Offset	Parameter Name	Byte Length	Description
2	RETCODE (hex)	1	Set by the NIC upon return. Valid return codes are: ■ 00h = Operation completed successfully ■ 01 = Invalid command code ■ 04 = NIC closed; it should be open
3		3	Reserved.
6	LOG_DATA	14	14 bytes of data sent by the NIC.

(2 of 2)

Table 35 Error Counters Available Through Read.Log

Byte	Meaning
0	Line errors (including LLC and MAC frame transmit CRC errors, and MAC frame receive CRC errors)
1	Internal errors
2	Burst errors
3	A/C errors
4	Abort delimiters
5	Reserved
6	Lost frames
7	Congestion errors (LLC and MAC frame receive overrun errors)
8	Frame copied errors
9	Frequency errors
10	Token errors
[11:13]	Reserved

Request.Interrupt

The Request.Interrupt command forces a MAC ASIC interrupt. It has no effect on ring communications. The NIC must be initialized but need not be opened.

Table 36 Request.Interrupt Command Parameters

Offset	Parameter Name	Byte Length	Description
0	COMMAND	1	Hex code = 00, Request.Interrupt
1		1	Reserved.
2	RETCODE (hex)	1	Set by the NIC upon return. Valid return codes are: ■ 00h = Operation completed successfully ■ 01 = Invalid command code

Restore.Open.Parms

The Restore.Open.Parms command modifies the OPEN_OPTIONS set by the Open.NIC command. The wrap option, remote program load, and modified token release bits are ignored.

Table 37 Restore.Open.Parms Command Parameters

Offset	Parameter Name	Byte Length	Description
0	COMMAND	1	Hex code = 02, Restore.Open.Parms.
1		1	Reserved.
2	RETCODE (hex)	1	Set by the NIC upon return. Valid return codes are: ■ 00 = Operation completed successfully ■ 01 = Invalid command code ■ 04 = NIC closed; it should be open
3		1	Reserved.
4	OPEN_OPTIONS	2	See Table 29.

Set.Funct.Address

The Set.Funct.Address command sets the functional address for the NIC to receive messages. If the FUNCT_ADDRESS field contains all zeros, any previously set functional address is disabled. Bits 31, 1, and 0 are ignored. The NIC accepts this command any time between when it is opened and when it is closed. The upper two bytes of the address are set to C000h.

Table 38 Set.Funct.Address Command Parameters

Offset	Parameter Name	Byte Length	Description
0	COMMAND	1	Hex code = 07, Set.Funct.Address.
1		1	Reserved.
2	RETCODE (hex)	1	Set by the NIC upon return. Valid return codes are: ■ 00 = Operation completed successfully ■ 01 = Invalid command code ■ 04 = NIC closed; it should be open
3		1	Reserved.
6	FUNCT_ADDRESS	4	New functional address to set.

Set.Group.Address

The Set.Group.Address command sets the group address for the NIC to receive messages. The NIC accepts this command any time between when it is opened and when it is closed. The upper two bytes of the address are set to C000h.

Table 39 Set.Group.Address Command Parameters

Offset	Parameter Name	Byte Length	Description
0	COMMAND	1	Hex code = 06, Set.Group.Address.
1		1	Reserved.

(1 of 2)

Table 39 Set.Group.Address Command Parameters (continued)

Offset	Parameter Name	Byte Length	Description
2	RETCODE (Hex)	1	Set by the NIC upon return. Valid return codes are: ■ 00 = Operation completed successfully ■ 01 = Invalid command code ■ 04 = NIC closed; it should be open
3		1	Reserved.
6	GROUP_ADDRESS	4	New group address to set.

(2 of 2)

Set.Multicast.Mode

The Set.Multicast.Mode command tells the NIC to enable or disable the multicast function. The NIC must be opened before this command is issued. See Table 40.

Table 40 Set.Multicast.Mode Command Parameters

Offset	Parameter Name	Byte Length	Description
0	COMMAND	1	Hex code = 0C, Set.Multicast.Mode.
1		1	Reserved.
2	RETCODE (hex)	1	Set by the NIC upon return. Valid return codes are: ■ 00 = Operation completed successfully ■ 01 = Invalid command code ■ 04 = NIC closed, should be open ■ 06 = Options invalid
3		1	Reserved.
4	MULTICAST_FLAG	1	Enable or disable multicast function. The driver should specify one of the following options: ■ 0 = Disable the multicast function ■ FF = Enable the multicast function

Set.Receive.Mode

The Set.Receive.Mode command tells the NIC to set different receive modes. This command must be issued after the NIC is opened. See Table 41.

Table 41 Set.Receive.Mode Command Parameters

Offset	Parameter Name	Byte Length	Description
0	COMMAND	1	Hex code = 1F, Set.Receive.Mode.
1		1	Reserved.

(1 of 2)

Table 41 Set.Receive.Mode Command Parameters (continued)

Offset	Parameter Name	Byte Length	Description
2	RETCODE (hex)	1	Set by the NIC upon return. Valid codes are: ■ 00 = Operation completed successfully ■ 01 = Invalid command code ■ 04 = NIC closed; should be open ■ 06 = Options invalid
3		1	Reserved.
4	RECEIVE_OPTIONS	2	See Table 42.

(2 of 2)

Table 42 Receive Options

Code	Meaning
0000	Normal mode. The NIC receives LLC and MAC frames that pass the address filter, and it forwards all received LLC frames to the host. The NIC does not forward received MAC frames to the host unless bits 15 or [4:3] in the OPEN_OPTIONS parameter of the Open.NIC command are set when the NIC is opened.
0002	The NIC receives only MAC frames that pass the address filter. It does not receive LLC frames. The NIC does not forward received MAC frames to the host unless bits 15 or [4:3] in the OPEN_OPTIONS parameter of the Open.NIC command are set when the NIC is opened.
0004	Promiscuous mode. The NIC receives all LLC and MAC frames regardless of their destination addresses. The NIC forwards all received LLC and MAC frames to the host, even if bits 15 or [4:3] in the OPEN_OPTIONS parameter of the OPEN.NIC command are not set when the NIC is opened.
0006	Promiscuous mode for LLC frames only. The NIC receives only those MAC frames that pass the address filter. It forwards all received LLC frames to the host. The NIC does not forward received MAC frames to the host unless bits 15 or [4:3] in the OPEN_OPTIONS parameter of the OPEN.NIC command are set when the NIC is opened.

Set.Sleep.Mode

The Set.Sleep.Mode command notifies the NIC that the host has entered the remote wake-up mode. See Table 43.

Table 43 Set.Sleep.Mode Command Parameters

Offset	Parameter Name	Byte Length	Description
0	COMMAND	1	Hex code = 05, Set.Sleep.Mode.
1		1	Reserved.
2	RETCODE (hex)	1	Set by the NIC upon return. Valid return codes: ■ 00 = Operation completed successfully ■ 01 = Invalid command code ■ 04 = NIC closed, should be open ■ 06 = Invalid options

(1 of 2)

Table 43 Set.Sleep.Mode Command Parameters (continued)

Offset	Parameter Name	Byte Length	Description
3		1	Reserved.
4	CAPABILITY	2	Magic/pattern: ■ Packet magic matching ■ Packet pattern matching ■ Magic and pattern packet matching

(2 of 2)

The NIC must be opened before the Set.Sleep.Mode command is issued. If the frame pattern capability is applied, the driver must issue the Change.Wakeup.Pattern command before issuing this command.

When the NIC receives the Set.Sleep.Mode command, it generates an SRB response to the host and then enters into the sleep mode, in which it behaves as follows:

- Remains inserted in the ring but does not report ring status change to the host.
- Does not forward MAC frames to the host.
- Ignores SRB commands and gives no SRB response to the host.
- Disables transmission of LLC frames.
- Redirects incoming LLC frames to be received into the local SRAM MAC buffer. It does not forward received LLC frames. Instead, the NIC compares received LLC frames to see if they match either the magic frame or the frame pattern specified by the driver. If there is a match, the NIC generates an interrupt to wake up the host. If there is no match, the NIC discards the frame.
- After it wakes up the host, the NIC exits from the sleep mode and restores normal operation.

ARB Commands

The adapter request block (ARB) passes receive information or commands from the CP to the host. It is located at absolute physical address D08A0h of the NIC SRAM. The ARB address is returned in the ARB_ADDRESS field of the open completion response SRB (see Table 30).

Table 44 summarizes the ARB commands.

Table 44 ARB Commands

Command Name	Code (Hex)	Description
Received.Data	81	Forwards MAC frames to the host.
Ring.Status.Change	84	Indicates a change in network status to the host.

The NIC issues ARB commands and the driver acts upon them. The NIC first prepares the ARB command, which is located at offset D08A0h of the SRAM. It then sets the arbc bit in the IntStatus register, which generates an interrupt to the driver.

To read ARB commands, the driver sets up the MacAccessCmd register with opCode = PrivateMemRead and localAddress = D08A0h. Then it reads from the MacData register. The read operation can be in byte or word lengths. The NIC does not automatically increment the localAddress after each read access. To read the next byte or word, the driver must write the MacAccessCmd register so that the localAddress points to the next location to be accessed.

Received.Data

The Received.Data command informs the host that data for a particular station has been received. The data must be moved from the receive buffers in shared memory and placed into host memory buffers. The Received.Data command provides the addresses of the buffers to the host in the ARB. In the last (or only) buffer containing the frame, bytes 2 and 3 contain 0000h; otherwise they contain the address of the next buffer plus two bytes.

When the host finishes processing the Received.Data command, it provides a return code in the ASB and interrupts the NIC. If the return code is 20 and the frame was an I frame destined for a link station, the NIC sets the local busy state for the link station. The host software must reset the local busy state when buffers become available. See Table 45 for Received.Data command parameters and Table 46 for ASB response parameters from the host.

Table 45 Received.Data Command Parameters

Offset	Parameter Name	Byte Length	Description
0	COMMAND	1	Hex code = 81, Received.Data.
1		3	Reserved.
4		2	Reserved.
6	RECEIVE_BUFFER	2	Offset to the receive buffer in shared memory (points to the MAC frame location).
8		1	Reserved.
9		1	Reserved.
10	FRAME_LENGTH	2	Length of the entire frame.
12	NCB_TYPE	1	Category of the message received. Hex values of the categories are: ■ 02 = MAC frame ■ 04 = I frame ■ 06 = UI frame ■ 08 = XID command poll ■ 0A = XID command not poll ■ 0C = XID response final ■ 0E = XID response not final ■ 10 = TEST response final ■ 12 = TEST response not final ■ 14 = Other or unidentified

Table 46 Received.Data ASB Response Parameters from the Host

Offset	Parameter Name	Byte Length	Description
0	COMMAND	1	Hex code = 81, Received.Data.
1		1	Reserved.
2	RETCODE (hex)	1	Set by the NIC upon return. Valid return codes to the MAC ASIC: ■ 00 = Operation completed successfully ■ 20 = Lost data on receive (no buffers available). Local busy is set if the NCB_TYPE is I frame. Valid return codes to the host: ■ FF = Response valid; ASB available ■ 01 = Unrecognized command code ■ 26 = Unrecognized command correlator; the receive buffer address is not that which the NIC expects ■ 40 Invalid STATION_ID
3		1	Reserved.
4	STATION_ID	2	ID of the station receiving data.
6	RECEIVE_BUFFER	2	Offset to the address of the first receive buffer in shared memory.

Ring.Status.Change

The Ring.Status.Change command indicates a change in the network status to the host. The indicated status may be the same as the last status if the NIC had to wait for the ARB to become available. After the host reads the command information from the ARB, it interrupts the NIC to acknowledge receipt of the command and indicate that the NIC can reuse the ARB. This command requires no response. See Table 47.

Table 47 Ring.Status.Change Command Parameters

Offset	Parameter Name	Byte Length	Description
0	COMMAND	1	Hex code = 84, Ring.Status.Change.
1		5	Reserved.
6	NEW_STATUS	2	Current network status.

ASB Commands

The adapter status block (ASB) passes host responses to ARB commands to the CP. It is located at absolute physical address DFED0h of the NIC SRAM. The ASB address is returned in the ASB_ADDRESS field of the open completion response SRB (see Table 30).

Table 48 summarizes ASB responses.

Table 48 ASB Responses

Command Name	Code (Hex)	Description
Received.Data	81	The device driver has completed receiving a forwarded MAC frame and the buffer can be reused by the NIC.

To provide responses to the NIC, the device driver sets up the MacAccessCmd register with opCode = PrivateMemWrite and localAddress = DFED0h. It then writes the ASB contents to the MacData register. The write operation can be byte or word length. The NIC does not automatically increment the localAddress after each write access. To write the next byte or word, the driver must write the MacAccessCmd register so that the localAddress points to the next location to be accessed. After the ASB contents are written, the device driver interrupts the NIC by setting the rasb bit in the MISR register.

Initializing the NIC

Before opening the NIC, the driver must perform the following procedure to initialize it:

1. Reset the NIC.
2. Set the pmbarVisible bit in the CPAttention register. Read the Pmbar register.
   - If the cpHold bit is cleared, indicating that a flash ROM is installed, do nothing but wait for the initialization SRB response interrupt as described in step 3.
   - If the cpHold bit is set, download the microcode to the NIC RAM as described in “Downloading the Microcode” in this chapter, and then clear the cpHold bit.
3. The NIC generates an initialization SRB response to indicate that it has completed the power-on self-test and initialization process.
4. The driver receives the initialization SRB response interrupt and completes the initialization by setting the DnBurstThresh, UpBurstThresh, TxStartThresh, and DnPriReqThresh registers.

Detecting Ring Speed

The NIC communicates with the network at either 4 Mbps or 16 Mbps. The choice of speed is determined by the ringSpeed (bit 1) setting in the SwitchSettings EEPROM field. The speed must be set properly for the NIC to open onto the ring.

The driver can either read the proper ring speed setting from some configuration disk file or determine the speed automatically.

To detect the proper ring speed, the driver should try to open onto the ring at the speed determined by the current value of the ringSpeed bit. If the Open command succeeds, then the current setting is correct and the driver can proceed with initialization. If the command returns a RETCODE equal to 7 (see Table 30) and an OPEN_ERROR_CODE equal to 7 (ring beaconing), the driver should assume that the ringSpeed setting is incorrect. The driver should then read the SwitchSettings field from the EEPROM and toggle the ringSpeed bit value. Then the driver should write the EEPROM with the new SwitchSettings value and retry the Open.NIC command.

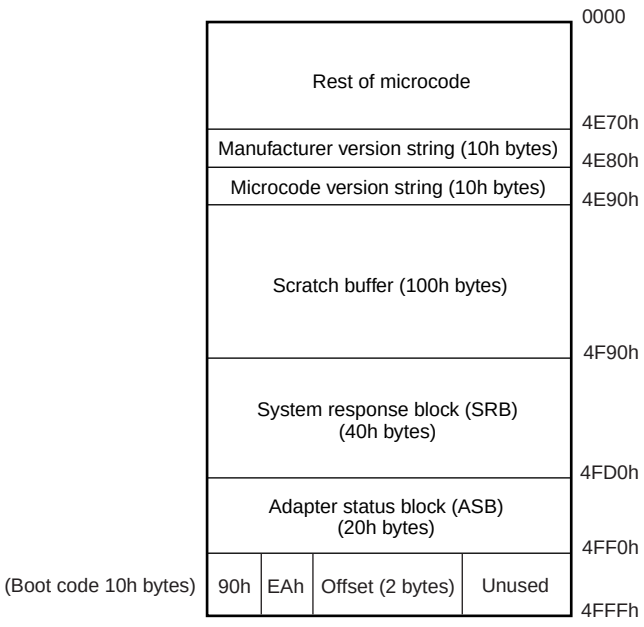
Downloading the Microcode

The procedures for downloading microcode with or without flash ROM installed are described in "Driver Configuration" in Chapter 4.

The driver can find the size of the microcode in the WILDFIRE.MAC file. The microcode size is a multiple of 16 bytes. The download starting address is calculated by subtracting the microcode size from the value 10000h. The driver should update the segment value of the boot code (4 bytes offset from the beginning of the boot code) and leave the offset value unchanged.

Assuming a WILDFIRE.MAC file size of 5000h, Figure 13 presents the format of the file. When the driver downloads the microcode file to the NIC memory, it should locate the end of 16 bytes (boot code) in the file and copy them to the address of FFF0h through FFFFh in the NIC's private memory.

Figure 13 WILDFIRE.MAC File Format



A

FRAME FORMAT

This appendix describes the token ring frame format. Although a complete description of the frame format is beyond the scope of this reference, some aspects are presented here because of their significance to receive error reporting and priority queueing.

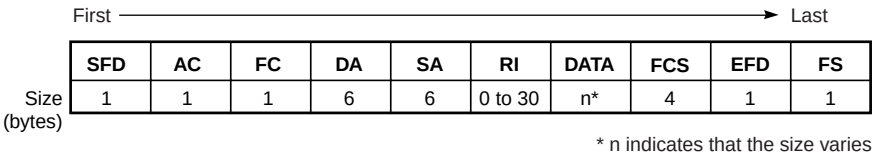
A token ring frame contains the fields listed in Table 49.

Table 49 Token Ring Frame Fields

Field	Abbreviation
StartFrameDelimiter	SFD
PhysicalControlField (access control [AC] and frame control [FC])	PCF
DestinationAddress	DA
SourceAddress	SA
SourceRoutingInformation	RI
Data	DATA
FrameCheckSequence	FCS
EndFrameDelimiter	EFD
PCF Extension Field	FS

The order in which the fields are transmitted over the ring is shown in Figure 14.

Figure 14 Ring Transmission Order



Bit Ordering

Bits within a byte are transmitted with the most-significant bit first. In all representations of data in this document, the most-significant bit is the left-most bit in the byte.

SFD and EFD Fields

The SFD field indicates the beginning of one of the following:

- Token ring frame
- Token
- Abort delimiter sequence (a contiguous pair of SFD and EFD fields)

The SFD field provides a unique definition for the bit and byte boundaries of every supported frame type.

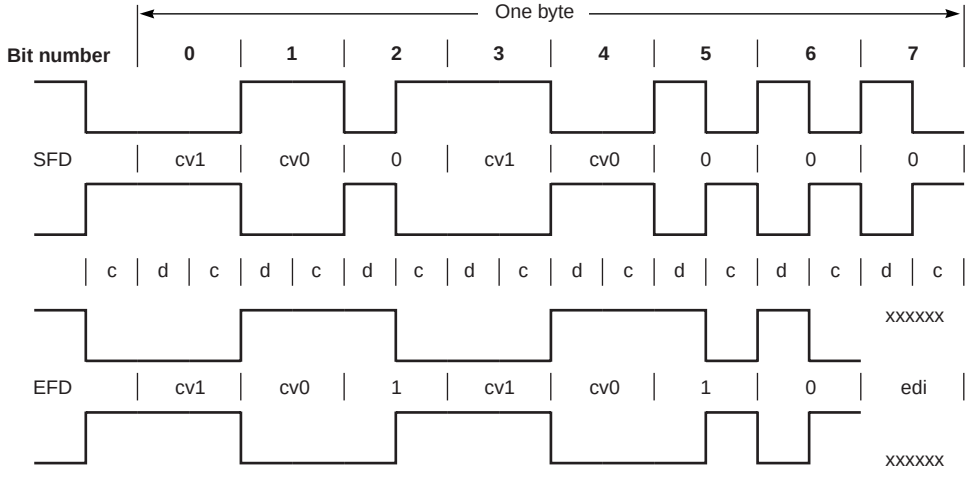
The EFD field indicates the end of a token ring frame, token, or abort delimiter sequence. To be valid, EFD fields must appear on bit and byte boundaries. Frame reception is successful only after a valid EFD is detected.

The MAC engine generates an EFD on byte boundaries when it ends the normal transmission of a frame. It generates an SFD when it issues a free token or begins a transmit immediate operation. It generates an abort delimiter sequence under the following conditions:

- The MAC engine is transmitting with a false free token (a corrupted token was captured and used for a frame transmission before the token corruption was detected).
- The transmitter underran.

If the MAC engine detects a code violation (cv) between the SFD and the EFD of a frame it is copying, it sets the error detect indicator of that frame (edi, bit 7 in the EFD) and indicates the error in bit 29 of the FrameStatus UPD entry. See Figure 15.

Figure 15 SFD and EFD Field Formats and Timing



cv = code violation
c = clock
d = data
edi = error detect indicator bit

The edi bit also indicates, when set, that a remote station detected an error in the frame. This bit is useful to management software for isolating possible error-prone segments of the ring. It is reported in bit 28 of the FrameStatus UPD entry.

AC Field

The AC field contains these bits:

- Three priority indicator bits (ppp)
- One token bit (t)
- One monitor count bit (m)
- Three reservation indicator bits (rrr)

AC Field Format

0	1	2	3	4	5	6	7
ppp			t	m	rrr		

The token ring handler supports eight priority levels of tokens, depending on the setting of the ppp and rrr bits. See Table 50.

Table 50 Token Priority Levels

ppp	bit 0	bit 1	bit 2	Priority
rrr	bit 5	bit 6	bit 7	(low to high)
	0	0	0	0 (normal)
	0	0	1	1
	0	1	0	2
	0	1	1	3
	1	0	0	4
	1	0	1	5
	1	1	0	6
	1	1	1	7

The driver must supply the AC field with packet data. The ppp bits [2:0] specify the priority level of the token that can be used to transmit the frame. For example, if ppp=101 (priority 5; see Table 50) tokens with priorities from 000 to 101 can be captured and used to transmit the frame. The MAC engine sets the remaining bits in the AC field according to ring protocol when the frame is transmitted.

The ppp bits specify the access priority only to the ring to which the station is connected. If the frame must hop to another ring, the priority that these bits convey is lost. To carry frame priority across the entire token ring network, the frame priority must also be specified in the FC field, described next.

FC Field

The FC field contains these bits:

- Two frame type bits (ff)
- Three reserved bits
- Three frame priority bits (yyy)

FC Field Format

0	1	2	3	4	5	6	7
ff		0	0	0	yyy		

FC Field Bit Descriptions

Bit	Name	Description
[1:0]	frameType	These bits specify whether the frame is a MAC or an LLC frame. The driver should only be concerned with LLC frames, and thus should set these bits to 2'b10.
[4:2]	Reserved	These bits have no function and should be set to 3'b000.
[7:5]	framePriority	These bits set the frame priority. The priority levels are the same as specified in the AC field (see Table 50).

A driver should set the frame priority to the level it wishes to attach to the frame and set the indicator priority in the AC field to the same value. This allows the frame's priority to be carried unaltered across the network.

These following priorities are recognized:

- 000 = standard, nonpriority LLC traffic
- 100 = bridge LLC traffic
- 101 = video LLC traffic
- 110 = audio LLC traffic
- 111 = reserved for MAC frames

DA Field

The DA field identifies the stations (destination addresses) for which the frame is intended. The DA field is 6 bytes (48 bits) long. Individual, group, broadcast, null, and functional address types are encoded. Station addresses may be administrated either locally or globally.

SA Field

The SA field is 6 bytes long, and contains the source address of the frame's originating station. Unlike a destination address, a source address is constrained to a single address. This implies that the most-significant bit of the frame address, the i/g bit, is 0. In place of the i/g bit is the routing information indicator bit (rii), which indicates the presence or absence of a routing information field (RI) in the frame.

RI Field

Frames may be routed through devices (known as bridges) from one ring to another. The RI field specifies the precise route through one or more bridges to the destination ring. The presence of an RI field is indicated when the rii bit of the SA field is set to 1.

The detailed structure of the RI field is described in these specifications:

- *ISO/IEC 10038, 1993*
- *ISO/IEC 8802-2, 1994*

DATA Field

The DATA field contains the contents of the frame. The length of the DATA field varies from 0 to *n*, where *n* is determined by the maximum frame size allowed (4,550 bytes at 4 Mbps and 18,200 bytes at 16 Mbps).

FCS Field

Token ring technology uses the following polynomial to generate the FCS field:

$$g(X) = X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^{8} + X^{7} + X^{5} + X^{4} + X^{2} + X^{1} + X^{0}$$

The FCS is calculated bit by bit starting with bit 0 in the FC field and ending with the last data bit. This occurs while the frame is being transmitted onto the token ring. Similarly, the FCS remainder of the protected field is accumulated bit by bit while the frame is being received from the ring. Before accumulating either FCS remainder, the accumulator is preset to 0xFFFFFFFF.

Each bit of the FCS remainder for the protected field is inverted to derive the transmitted FCS field.

To check a received frame for errors, the FCS remainder for the received vector is accumulated along with the data. If no error is present, the result is the constant 0xC704DD7B.

FS Field

The FS field contains two bits of interest:

- ar = address recognized
- fc = frame copied

The originating station sends the ar and fc bits as 0. Another station sets the ar bit if it recognizes the destination address as its own. The station also sets the fc bit if it copies the frame.

B

ERRATA LIST AND SOFTWARE SOLUTIONS

This appendix describes 3C359 NIC anomalies and their software solutions.

Hash Calculation

Hashing is performed by accumulating a CRC on the frame's DA field if the DA field's most-significant bit is a 1 (indicating a group frame). The polynomial used for the CRC calculation is:

$$g(X) = X^{32} + X^{26} + X^{23} + X^{22} + X^{16} + X^{12} + X^{11} + X^{10} + X^{8} + X^{7} + X^{5} + X^{4} + X^{2} + X^{1} + X^{0}$$

The remainder is preset to all ones, and it is then modified by dividing the DA field by the polynomial. The hash function is the low-order six bits (the coefficients of X^5 to X^0) of the remainder. Normally this operation should be done beginning with the most-significant bit (0) of the DA field and ending with the least-significant bit (47). The order of the calculation is inconsequential in itself except that the driver needs to calculate the hash in exactly the same way that the hardware does in order to set the hash table bits correctly.

Most software routines that are commonly available to do this calculation were originally written for Ethernet, and they were written to operate on a stream of bytes (not bits). The bit order in Ethernet is not the same as it is for token ring technology. Within each token ring byte, the bit order as it appears on the wire is reversed. So to use an algorithm that was created for Ethernet, the bits within each byte must be reversed before the hash is calculated.

However, a bug in the 3C359 NIC silicon causes the bits within each byte to be calculated in this order: 3, 2, 1, 0, 7, 6, 5, 4. Therefore, an Ethernet algorithm being used for token ring must swap the nibbles within each byte (but not do any bit swapping) before performing the calculation. The sample code shown below demonstrates this technique.

The least-significant six bits of the resulting remainder are used as a vector into the hash table. The driver should set those bits in the tables that correspond to each address of interest. After the 3C359 NIC has performed the same calculation on multicast frames, it checks the state of the bit at the addressed location. If the addressed bit is set, the frame is uploaded. If the addressed bit is clear, the hardware discards the frame.

```
*   CalculateCRC() - Calculate CRC based on input addr.
*
*       Input:  Addr = pointer to a string containing 6-byte addr
*       Return: CRC = double word CRC value
************************************************************/
ULONG CalculateCRC(BYTE *Addr)
{
    const ULONG Poly=0x04c11db6L;
```

```
    ULONG CRCValue=0xffffffffL,
          CurrentCRCHigh;
    UINT  NumberOfBytes=6,
          CurrentBit;
    BYTE  CurrentByte;

    for (; NumberOfBytes; NumberOfBytes--)
    {
        CurrentByte = SwapNibble(*Addr);        // swap the nibbles
        Addr++;
        for (CurrentBit=8; CurrentBit; CurrentBit--)
        {
            CurrentCRCHigh = CRCValue>>31;
            CRCValue <<= 1;
            if(CurrentCRCHigh^(CurrentByte&0x01))
            {
              CRCValue ^= Poly;
              CRCValue |= 0x00000001L;
            }
            CurrentByte >>= 1;
        }
    }
    return CRCValue;
}   // end CalculateCRC()


/*****************************************************************
*   SwapNibble() - Swap nibbles in a byte, so that:
*                   bit 7 6 5 4 3 2 1 0
*           becomes bit 3 2 1 0 7 6 5 4
*****************************************************************/
BYTE SwapNibble(BYTE b)
{
    asm {
        mov     al,b
        mov     ah,al
        and     ah,0f0h
        and     al,0fh
        shr     ah,4
        shl     al,4
        or      al,ah
    }
    return _AL;
}   // SwapNibble()
```

INDEX OF REGISTERS

INDEX OF BITS

INDEX

R

S

T

U

W