Personal Computer
Hardware Reference
Library

RT PC Technical Reference Token-Ring Network Adapter

SK2T-0291-1

Personal Computer
Hardware Reference
Library

RT PC Technical Reference
Token-Ring Network Adapter

Second Edition (February 1987)

Changes are made periodically to the information herein; these changes will be incorporated in new editions of this publication.

Federal Communications Commission (FCC) Statement

About This Book

Purpose

This manual describes the operating characteristics of the IBM Token-Ring Network RT PC Adapter.

Audience

The information in this publication is for reference, and is intended for hardware and program designers, programmers, engineers, and anyone else who needs to understand the design and operation of the IBM Token-Ring Network RT PC Adapter.

How to Use This Book

The manual consists of four chapters: Chapter 1 is an introduction to the function and features of the IBM Token-Ring Network RT PC Adapter, Chapter 2 describes the hardware and its operation, Chapter 3 describes the software required to utilize the Adapter, Chapter 4 provides the environmental requirements to ensure proper operation. Appendix A is provided for example software.

Related Publications

Refer to the following IBM publications for additional information:

- *A Building Planning Guide for Communication Wiring* G230-8059
- *IBM Cabling System Planning and Installation Guide,* GA27-3361
- *Using the IBM Cabling System with Communication Products,* GA27-3677.

Prerequisite Publications

Information from the following publications is necessary for use of this manual.

- *IBM Token-Ring Network Architecture Reference*
- *IEEE Standards for Token-Ring Networks: Logical Link Control, ANSI/IEEE Std 802.2 - 1985*

Contents

Figures

Tables

Chapter 1. Overview

CONTENTS

Description

The IBM Token-Ring Network, a token-ring, star-wired network, can accomodate up to 260 attaching devices (printers, processors, controllers). These work stations, within a moderate-sized geographic area, can be connected to one another via a series of cable, multistation access units, and special adapter cards installed in the attaching devices. Application programs running in each work station (such as an IBM RT Personal Computer) can direct the adapter to become a part of the ring.

Components of the IBM Token-Ring Network

Cable connects work areas to wiring closets and wiring closets to one another.

Access units, such as IBM 8228 Multistation Access Units, are connected to the ring with patch cables. You may connect up to eight attaching devices, such as IBM RT Personal Computers, to each 8228 Access Unit.

Adapter cards, such as the IBM Token-Ring Network RT PC Adapters, are installed in each attaching device to enable the devices to communicate to one another on the network.

Communicating on the IBM Token-Ring Network

The IBM Token-Ring Network uses a protocol to control data flow, monitor ring conditions, and encapsulate and route data for devices attached to the physical ring. The Token, Frame and Medium Access Control (MAC) are involved in this LAN protocol. This protocol is not totally explained in this manual. See *IBM Token-Ring Network Architecture Reference* for more detail about this protocol.

Adapter Communication on the Network

An IBM RT PC is ready to become part of the network when:

- An IBM Token-Ring Network RT PC Adapter has been installed.
- The IBM Token-Ring Network RT PC Adapter is connected to the IBM Cabling System using an attachment cable to a multistation access unit. (The Token-Ring Adapter can also be attached to telephone twisted-pair media by using the Type 3 Media Filter. Reference the *IBM Token-Ring Network Telephone Twisted-Pair Media Guide*, GA27-3714.)
- A program to support the IBM Token-Ring Network RT PC Adapter has been loaded into the RT PC.

To communicate on the network a device adapter obtains a token and changes the token to a frame containing control information and data. The frame is then passed around the ring, and each device on the ring monitors the frame and retransmits it. When the frame reaches the device to which it is addressed, the device copies the frame while retransmitting it and indicates in the frame that the data has been received. The frame continues in the same direction around the ring until it returns to the device that sent the frame. The sending device removes the frame from the ring replacing it with a token.

Transmission of Data on the Network

When the RT Personal Computer is powered on, the Token-Ring Network Adapter goes through a series of operational diagnostics before attempting to attach to the network. The adapter card verifies:

- The correct operation of the adapter card.
- That a cable between the attaching device and the 8228 Access Unit exists.
- That a test message sent along the lobe cable (the cable between the attaching device and the 8228 Access Unit) can pass through the cable and return unchanged.

The **ADAPTER.OPEN** command tells the adapter to send a direct current to the 8228 Access Unit to which it is attached. This causes the attaching device to become an active part of the ring, receiving, monitoring, and retransmitting ring traffic.

The device places data on the ring by changing a token into a frame that will pass around the ring in place of the token.

Token Format

A token is a unique sequence of bits in 3 bytes transmitted on the ring. The sequence is a start delimiter followed by an access control field of one byte and ending with an end delimiter byte. Each byte is eight bits in length without parity. Both the start delimiter (SD) and the end delimiter (ED) are unique bit arrangements recognized by all adapters compatible with an IBM Token-Ring Network. The access control (AC) field controls access to the ring. Refer to Figure 1-1.

SD 1 Byte	AC 1 Byte	ED 1 Byte

Figure 1-1. Token Format

The same three bytes are included in a frame. However, the token indicator bit is set to 1 in the AC byte to indicate a frame rather than a token. Additional information bytes are included between the AC byte and the ED byte.

Frames

A frame consists of a physical header followed by information bytes and a physical trailer. Refer to figure 1-2.

The physical header consists of the starting delimiter (SD) byte, the access control (AC) byte with the token indicator bit set to 1, the frame control (FC) byte, 6 bytes of destination address, 6 bytes of source address, and from zero to 18 bytes of routing information. This is followed by the information field which contains the user-provided data. At the end of the frame is the physical trailer which consists of a 4 byte frame check sequence (FCS), the ending delimiter (ED) byte, and the frame status (FS) byte. The data contained in the frame may be of two types:

- MAC frames
- Non-MAC frames

Medium Access Control (MAC) frames contain information about the status of an adapter or the ring itself. Non-MAC frames contain data and messages that users transmit to one another. The 2 most significant bits of the FC byte define the frame type. The types are :

B'00'= MAC frame
B'01'= LLC frame
B'10'= reserved
B'11'= reserved

Frames are described in detail in the *IBM Token-Ring Network Architecture Reference.*

SD	**AC**	**FC**	**Dest. Addr.**	**Source Addr.**	**ROUTING**	**Info Field**	**FCS**	**ED**	**FS**
1 byte	1 byte	1 byte	6 bytes	6 bytes	0-18 bytes	---- bytes	4 bytes	1 byte	1 byte

Figure 1-2. Frame Format

Chapter 2. Description

CONTENTS

Architecture

The Adapter hardware can be conceptualized as three different interfaces: the Host System Interface, the Adapter Bus interface which provides the internal communication link of the adapter, and the Ring Interface which provides the communications interface to the IBM Token-Ring Network. See figure 2-1. The information which follows is presented in this manner with a hardware description of the Host System Interface, the Adapter Bus, and the Ring Interface. Schematic diagrams for the Adapter Card and some physical characteristics of the Adapter Card are also provided.

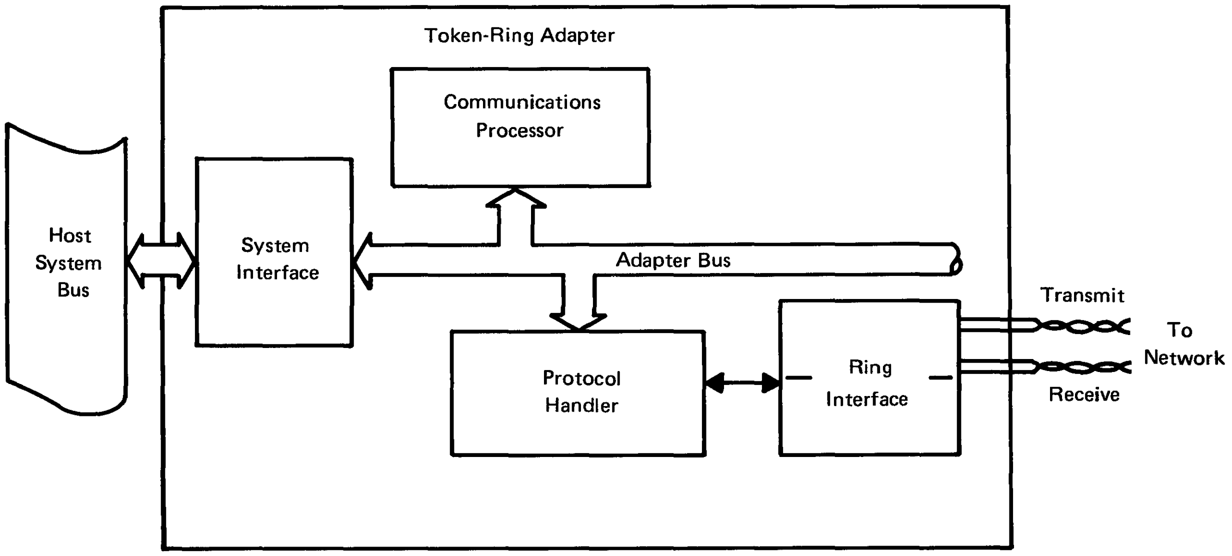


Figure 2-1. Token-Ring Adapter Architecture

Features

The RT PC Token-Ring Network Adapter is designed to provide a communications interface between an IBM Token-Ring Network and an Intel 80286-type interface such as the IBM RT Personal Computer. A summary of the Adapter's features are listed below:

- IEEE 802.5 Token-Ring LAN Compatible
- Jumper selectable DMA levels and I/O address space
- Microcode Update Capability -EPROM Sockets for Microcode Instructions (Jumper Selectable)
- 16K Bytes Data Buffer
- DMA Bus Master
- Direct I/O Addressing
- Interrupt sharing
- Jumper selectable* interrupt levels

* Not applicable to card assembly 56X2295 or 69X8139

System Interface Description

This interface provides for read/writes between the Token-Ring Network Adapter (hereafter referred to as Adapter) and the IBM RT PC, termed a Direct Input/Output (DIO), and Direct Memory Access (DMA) in which the Adapter is bus owner and controls read/writes to the Host System Bus. The interface also features a sharable interrupt on interrupt levels 11* and 12.

* Not applicable to card assembly 56X2295 or 69X8139.

Signal Levels

Signals are active high except when the signal name is preceded by the symbol "-", which indicates that the signal is active low.

IBM RT PC Bus Connector Pin Description

System Address Bus (SA0 - SA19, LA17 - LA23)

This tri-state bus consists of 27 latched address lines. This bus is driven by the Adapter when it is Bus Owner. During DIO cycles, address bits SA16 and LA17 - LA23 are all 0. LA23 is the most significant bit and SA0 is the least significant bit.

Address bits SA1 and SA2 are wired to bits SRS1 and SRS0, respectively, on the system interface of the Adapter Card. The state of SRS1 and SRS0 selects one of four halfword registers through the DIO interface. A summary of the register addresses is provided in the following table.

Table 2-1. DIO Register Addresses

SA2 (SRS0)	SA1 (SRS1)	BHE	Register Accessed
0	0	0	SIFD MSB and LSB
0	1	0	SIFD/INC MSB and LSB
1	0	0	SIFADR MSB and LSB
1	1	0	SIFCMD and SIFSTS

Note: When -BHE is 0, both bytes of the halfword register are accessed together; when -BHE is 1, the LSB of the halfword register is accessed.

System Bus High Enable (-SBHE)

This is a tri-state bidirectional line that is driven during a DMA operation and is an input during a DIO operation. It is placed in the high impedance state at all other times. The signal is active when the high byte is to be enabled on the output bus. As an input, the signal selects the most significant byte of the DIO register.

Address Enable (AEN)

Address enable is used in the address decode of the Adapter to ensure proper device selection during a DIO cycle. AEN low and a valid address properly selects the Adapter.

System Data Bus (SD0 - SD15)

This tri-state bidirectional bus consists of sixteen data lines which are used to transfer data between the Adapter and system memory. The Adapter sends data on write cycles and the memory sends data on read cycles. SD15 is the most significant bit and SD0 is the least significant bit.

Memory Read (-MEMR)

The Memory Read Command is a tri-state signal driven by the Adapter card during DMA operations. It instructs the selected device in the memory space to drive data onto the data bus.

Memory Write (-MEMW)

The Memory Write Command is a tri-state signal driven by the Adapter card during DMA operations. It instructs the selected device in the memory space to store the data present on the data bus.

I/O Read (-IOR)

The I/O Read command is a tri-state signal driven by the System Bus which instructs the Adapter card to drive data onto the data bus during DIO operations. This data is held on the bus for as long as -IOR is active.

I/O Write (-IOW)

The I/O Write command is a tri-state signal driven by the System Bus which instructs the Adapter card to store the data presently on the data bus during DIO operations.

I/O Channel Ready (I/OCHRDY)

I/O Channel Ready is a normally high, open collector signal that is used by the Adapter to lengthen a current cycle. During a DIO, when the Adapter is a controlled device, I/OCHRDY is driven low after a valid select is detected and held low until the Adapter is able to complete the cycle. During a DMA, when the Adapter controls the bus, the current cycle may be lengthened by the Host System Bus driving I/OCHRDY low.

I/O Card Select 16-Bit (-I/OCS16)

This signal is driven active by the Adapter card after decoding a valid address.

DMA Request (DRQ3*, DRQ5, DRQ6, DRQ7*)

The DMA Request line (DRQx, jumper selectable) is driven high by the Adapter to request DMA service or control of the system bus. DRQx is held high by the Adapter card throughout the bus operation. DRQ3 has the highest priority and DRQ7 has the lowest priority. The DRQ is tri-stated by the assertion of a System Reset. In the case of a soft reset, a DIO to the Adapter Reset address, the DRQ will be tri-stated and this may cause the DMA controller to erroneously detect a DMA request. To avert this problem the DMA controller should be disabled before a soft reset is asserted.

* Not applicable to card assembly 56X2295 or 69X8139.

DMA Acknowledge (-DACK3*, -DACK5, -DACK6, -DACK7*)

The DMA Acknowledge signal is driven active by the System Bus in response to the corresponding DMA request. The -DACKx signal indicates that the Adapter will be granted use of the bus following proper activation of the -MASTER control signal.

* Not applicable to card assembly 56X2295 or 69X8139.

Bus Master (-MASTER)

This signal is used by the Adapter to gain control of the I/O Bus after receiving a DMA Acknowledge. After receiving the -DACK the Adapter drives the -MASTER line low, then after a delay the Adapter drives the address and data lines. When releasing the bus all address, data, and control lines are tri-stated before driving -MASTER high (inactive).

Interrupt Request (IRQ11*, IRQ12)

The interrupt request line is used to signal the System processor that the Adapter card requires attention. An interrupt request is generated by pulsing the IRQxx line. This pulse is formed by a transition from high-to-low and held low for two clock cycles before transitioning high. The low-to-high transition of the pulse signals an interrupt to the Using Product System. A 2-kilohm pullup resistor is included on the output of the IRQxx tri-state driver to accommodate interrupt sharing. This driver has an enable/disable function which is programmable by the Using Product system software and an interrupt is disabled by a system reset.

* Not applicable to card assembly 56X2295 or 69X8139.

Reset Driver (RESET DRV)

Reset Driver is an input to the Adapter used to set internal devices to a known state.

I/O Pin Assignments

SIGNAL NAME	I/O PIN
SA0	A31
SA1	A30
SA2	A29
SA3	A28
SA4	A27
SA5	A26
SA6	A25
SA7	A24
SA8	A23
SA9	A22
SA10	A21
SA11	A20
SA12	A19
SA13	A18
SA14	A17
SA15	A16
SA16	A15
SA17	A14
SA18	A13
SA19	A12
AEN	A11
I/O CHRDY	A10
SD0	A09
SD1	A08
SD2	A07
SD3	A06
SD4	A05
SD5	A04
SD6	A03
SD7	A02
GND	B31
+5	B29
DRQ3*	B16
-DACK3*	B15
-IOR	B14

SIGNAL NAME	I/O PIN
-IOW	B13
GND	B10
+5	B03
RESET DRV	B02
GND	B01
SD15	C18
SD14	C17
SD13	C16
SD12	C15
SD11	C14
SD10	C13
SD9	C12
SD8	C11
-MEMW	C10
-MEMR	C09
LA17	C08
LA18	C07
LA19	C06
LA20	C05
LA21	C04
LA22	C03
LA23	C02
-SBHE	C01
GND	D18
-MASTER	D17
+5	D16
DRQ7*	D15
-DACK7*	D14
DRQ6	D13
-DACK6	D12
DRQ5	D11
-DACK5	D10
IRQ12	D05
IRQ11 *	D04
-I/OCS16	D02

* Not applicable to card assembly 56X2295 or 69X8139

Adapter I/O Addresses

Table 2-2. Adapter I/O Addresses

Primary (Hex Address)	Alternate (Hex Address)	Register
01C0	0140	SIF Data
01C2	0142	SIF Data INC
01C4	0144	SIF Address
01C6	0146	Command/Status
01C8	0148	Enable Interrupts
01CC	014C	Disable Interrupts
01CE	014E	Adapter Reset
06F4	06F4	Interrupt Level Enable (ILE), Level 12

Adapter Bus Interface

The Adapter Bus Interface is internal to the Adapter Card. It interfaces to the Communications Processor, the Protocol Handler, the System Interface, as well as the Adapter RAM and sockets for the Microcode Update EPROMs.

Adapter Bus Memory Map

The memory map for the Adapter is shown in Figure 2-2. This memory map illustrates a memory space of 64 Kbytes.

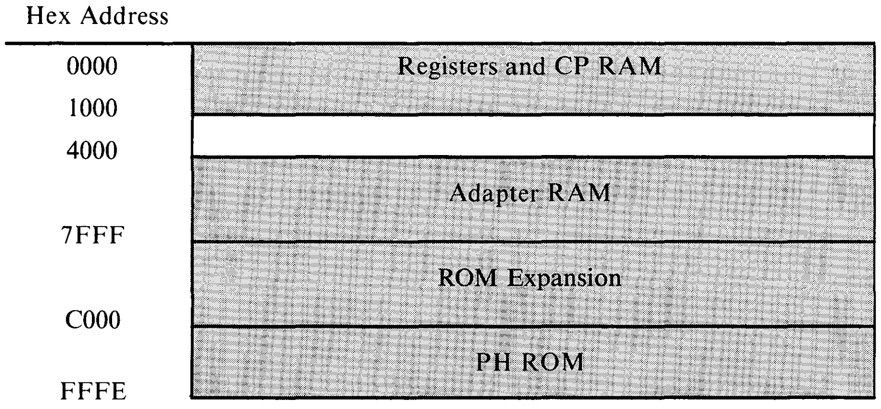


Figure 2-2. Adapter Bus Memory Map

Microcode Update EPROMs

The Adapter's control program currently resides in ROM within the Protocol Handler. The adapter features sockets for three EPROMs which allow for microcode updates.

Ring Interface

Hardware Description

The interface of the Adapter to the IBM Token-Ring Network is via the Ring Interface circuit, containing two bipolar MSI circuits and several discrete components. The two bipolar MSI chips are an anolog chip set which provide the IEEE 802.5 compatible interface functions between the Adapter's Protocol Handler (PH), and the IBM Token-Ring Network. These chips are referred to as the Adapter's Ring Transceiver and Ring Controller. The Transceiver provides the transmit and receive functions while the Controller provides ring interface control functions. Included in the schematic shown in Schematic Diagrams, are all the necessary passive components in addition to an Adapter Address and a Processor Activity Timer which are discussed in further detail below.

Adapter Address

The 1024x4 PROM and the 74LS164 in the schematic in Chapter 2, implement the Adapter Address feature of the Adapter. This feature provides the Adapter with a 48-bit node address if a node address is not passed during the **OPEN** command. Whenever the Adapter is initialized, the Communication Processor will fetch the Adapter Address from the address module. If the node address passed to the Adapter during the **OPEN** command is zero, the Adapter Address will be used as the Adapter's Ring Station Address. The Adapter software tests for the presence of the address module circuitry during initialization. The Adapter reads the PROM and checks the integrity of the data in the address PROM, but does not check for valid addresses. An error in the address PROM will cause the **OPEN** command to terminate with a node address error.

Notes:

1. On card assemblies 56X2295 and 69X8139, the Adapter Address feature is selectable through the use of Jumper J4. The jumper is set in either the internal or system position with the internal address as the default position.
2. On other card assemblies, the internal address is always selected (no jumper).

Processor Activity Timer

The Processor Activity Timer (PAT) feature includes a one-shot which is configured to provide a 20 millisecond pulse when a negative transition occurs on signal -PHNSRT. The inverted Q output of the one-shot serves as the active-low signal causing the Adapter to physically insert onto the ring. The Adapter's program code toggles the -PHNSRT line every 10 milliseconds whenever the Adapter is inserted on the ring. The failure of the Adapter to toggle the -PHNSRT causes the one-shot to time out and the Adapter is physically de-inserted from the ring.

Notes:

1. On card assemblies 56X2295 and 69X8139, this timer may be disabled with Jumper J5. The default position of Jumper J5 is enabled.
2. On other card assemblies, this timer is always enabled (no jumper).

Ring Connector Interface

Table 2-3. Ring Connector Pin Assignment

Pin Number	Pin Name
POS 1	Ring In 1 (Green)
POS 2	DC Common
POS 3	+ 5 Volts
POS 4	DC Common
POS 5	Ring Out 2 (Black)
POS 6	Ring In 2 (Red)
POS 7	DC Common
POS 8	DC Common
POS 9	Ring Out 1 (Orange)

Physical Signaling

Differential Manchester Code

The Token-Ring Protocol calls for a ring signaling format called Differential Manchester Code. This signaling scheme follows these rules:

1. A signal transition always occurs in the center of the bit time.
2. A zero bit has a transition at the beginning of the bit time. A one bit has no transition during this time.

Figure 2-3 illustrates this coding scheme. The signaling transitions are symmetric around the zero volt level, thus providing an average zero volt DC level. This facilitates transformer coupling of the Adapter's transmitter and receiver to the ring.

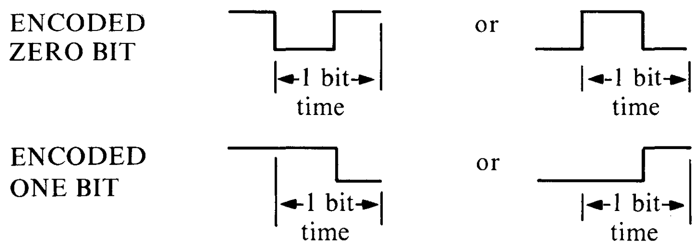


Figure 2-3. Differential Manchester Code

Ring Clocking

An Adapter, randomly designated via a claim-token process, provides master clocking to the ring by deriving its timebase from a crystal oscillator. This Adapter is called the Active Monitor. Any Adapter on the ring may assume the role of Active Monitor.

The remaining ring stations on the ring derive their timebase by phase synchronizing a voltage controlled oscillator to the incoming bit stream. This Phase Locked Loop (PLL) derived clock provides the necessary timebase from which the bit stream is received and transmitted by the Adapter.

For more information about the IBM Token-Ring Network operations see the *IBM Token-Ring Network Architecture Reference.*

Schematic Diagrams

The following section displays the logic diagrams for the RT PC Token-Ring Network Adapter.

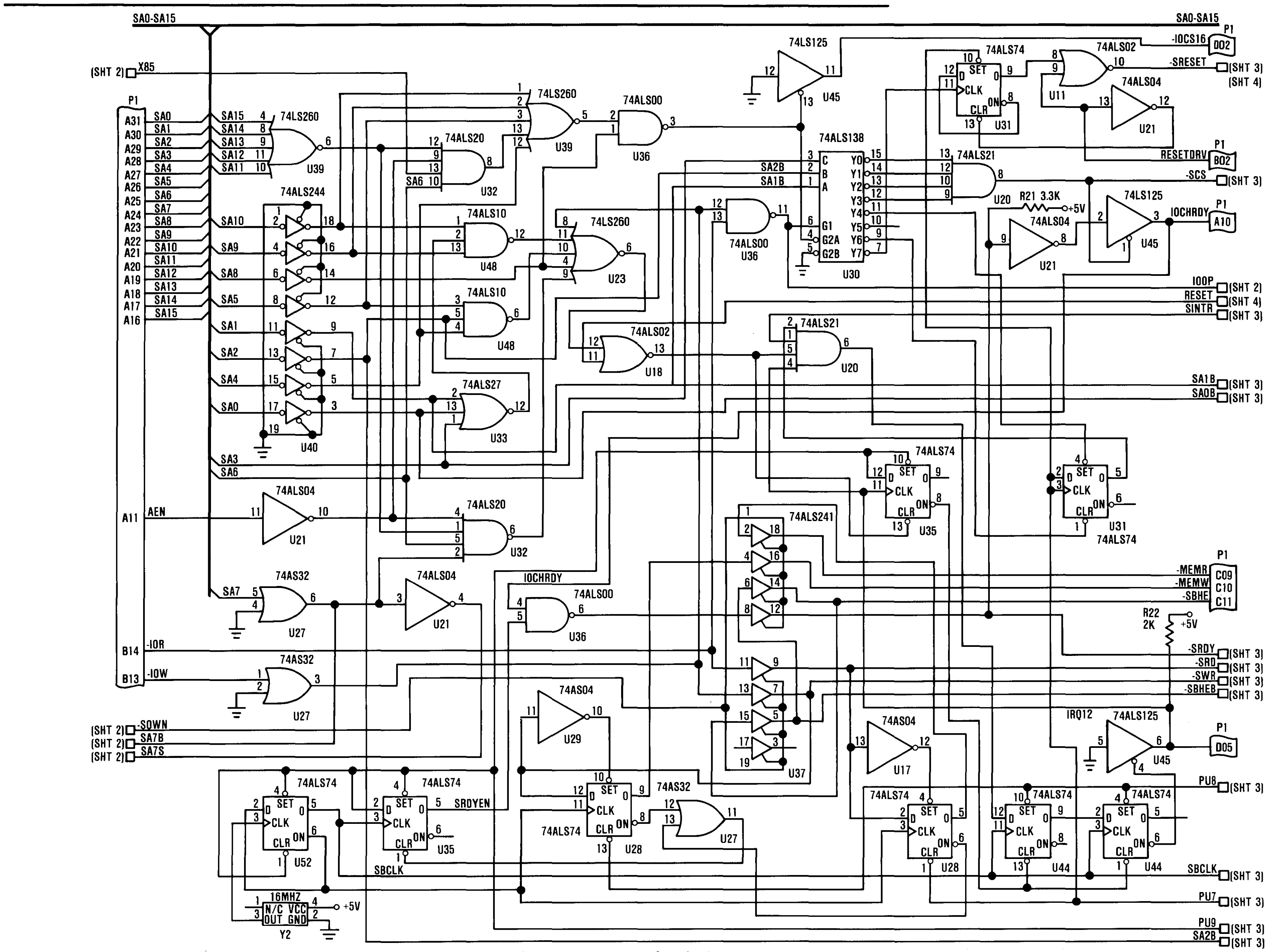


Figure 2-4A. Token-Ring Adapter Logic Diagram (card assembly 56X2295 or 69X8139) (Sheet 1)

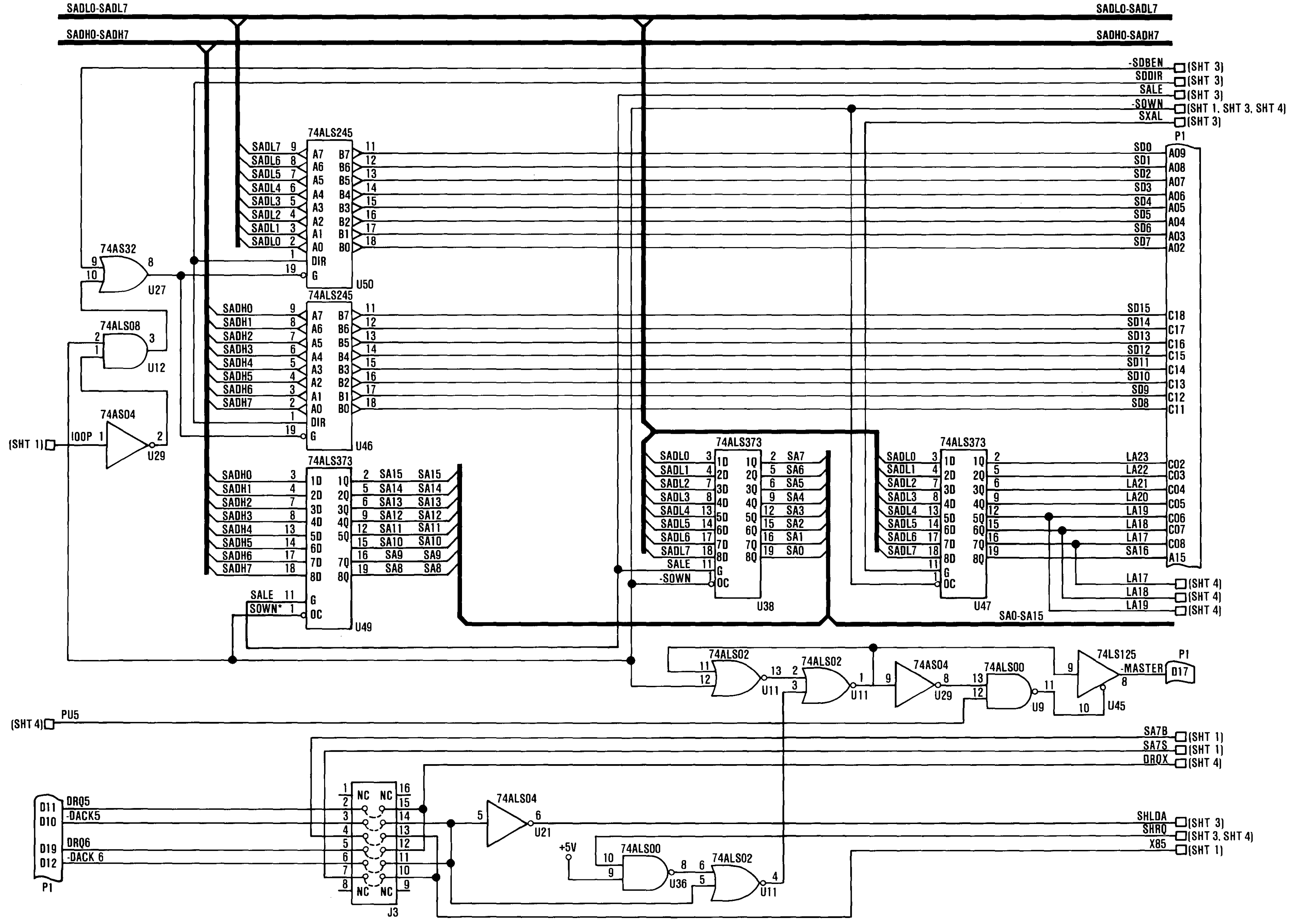


Figure 2-4A. Token-Ring Adapter Logic Diagram (card assembly 56X2295 or 69X8139) (Sheet 2)

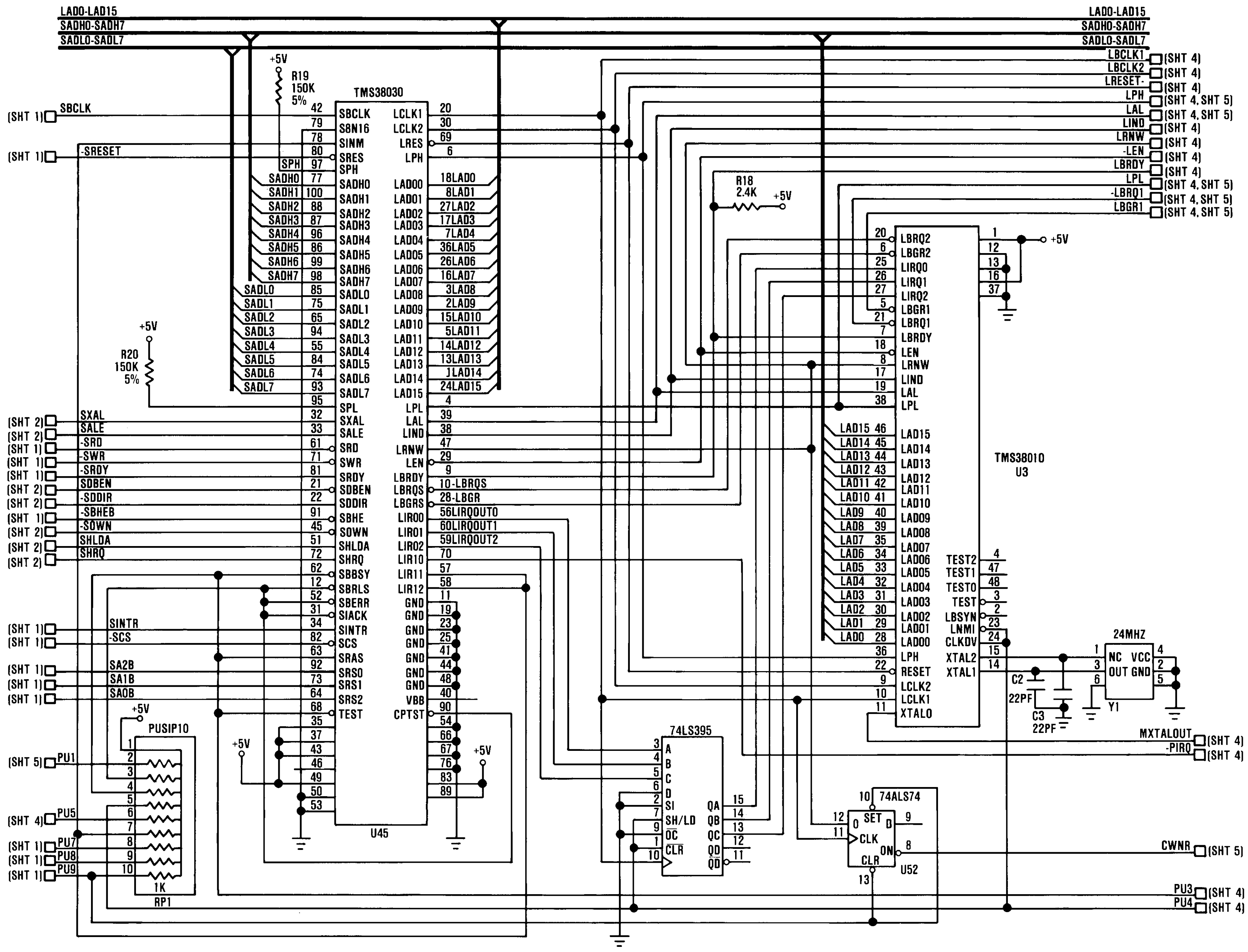


Figure 2-4A. Token-Ring Adapter Logic Diagram (card assembly 56X2295 or 69X8139) (Sheet 3)

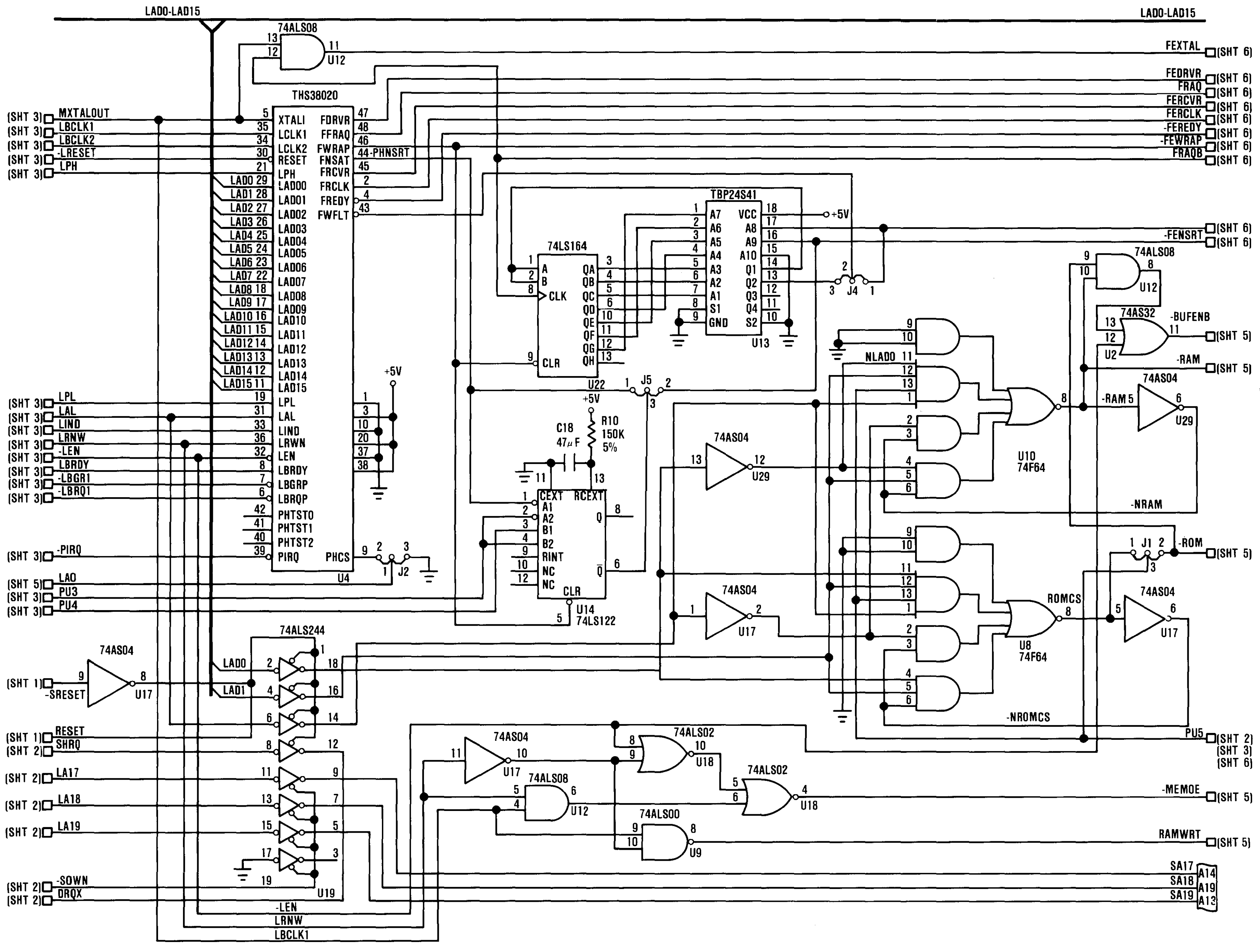


Figure 2-4A. Token-Ring Adapter Logic Diagram (card assembly 56X2295 or 69X8139) (Sheet 4)

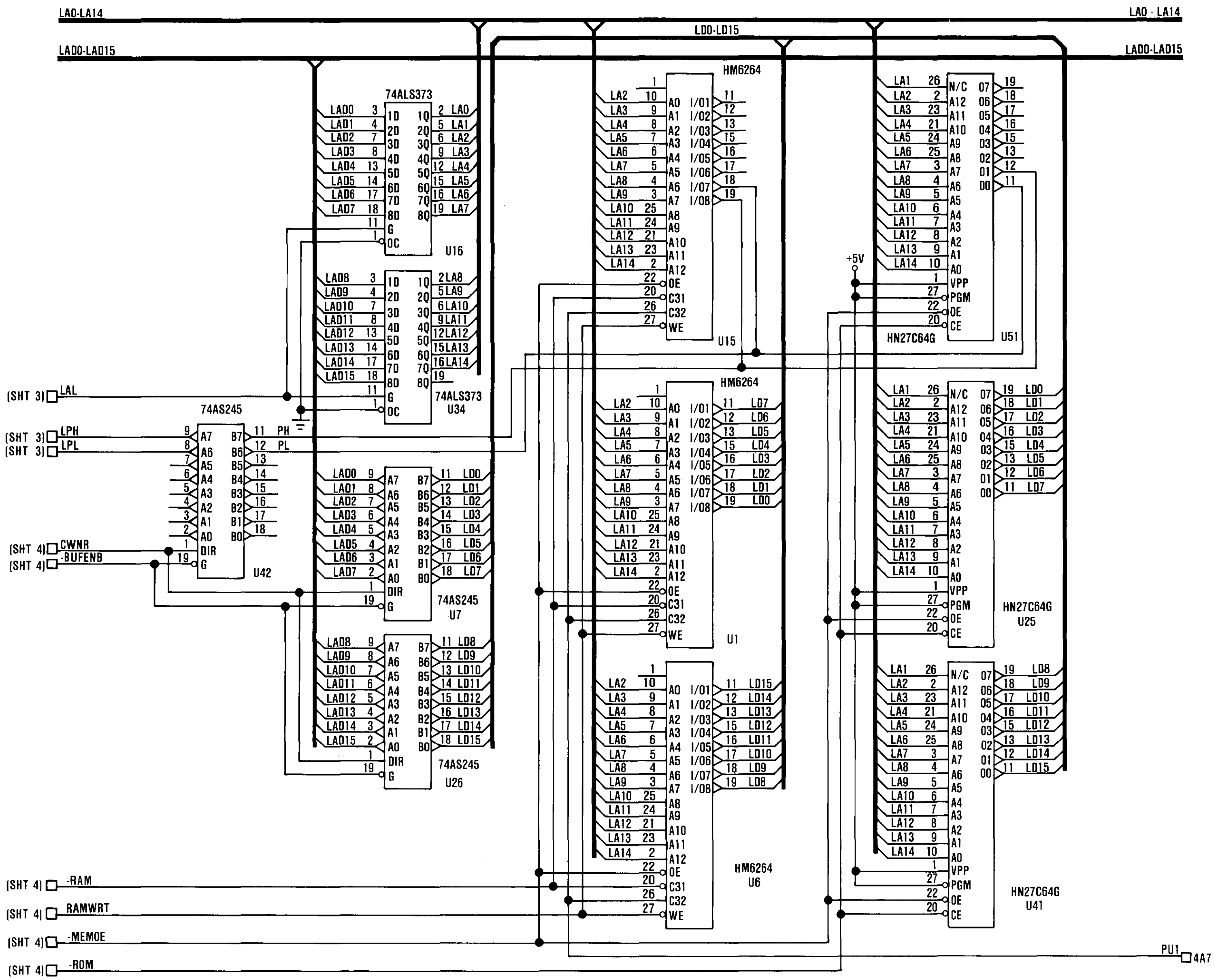


Figure 2-4A. Token-Ring Adapter Logic Diagram (card assembly 56X2295 or 69X8139) (Sheet 5)

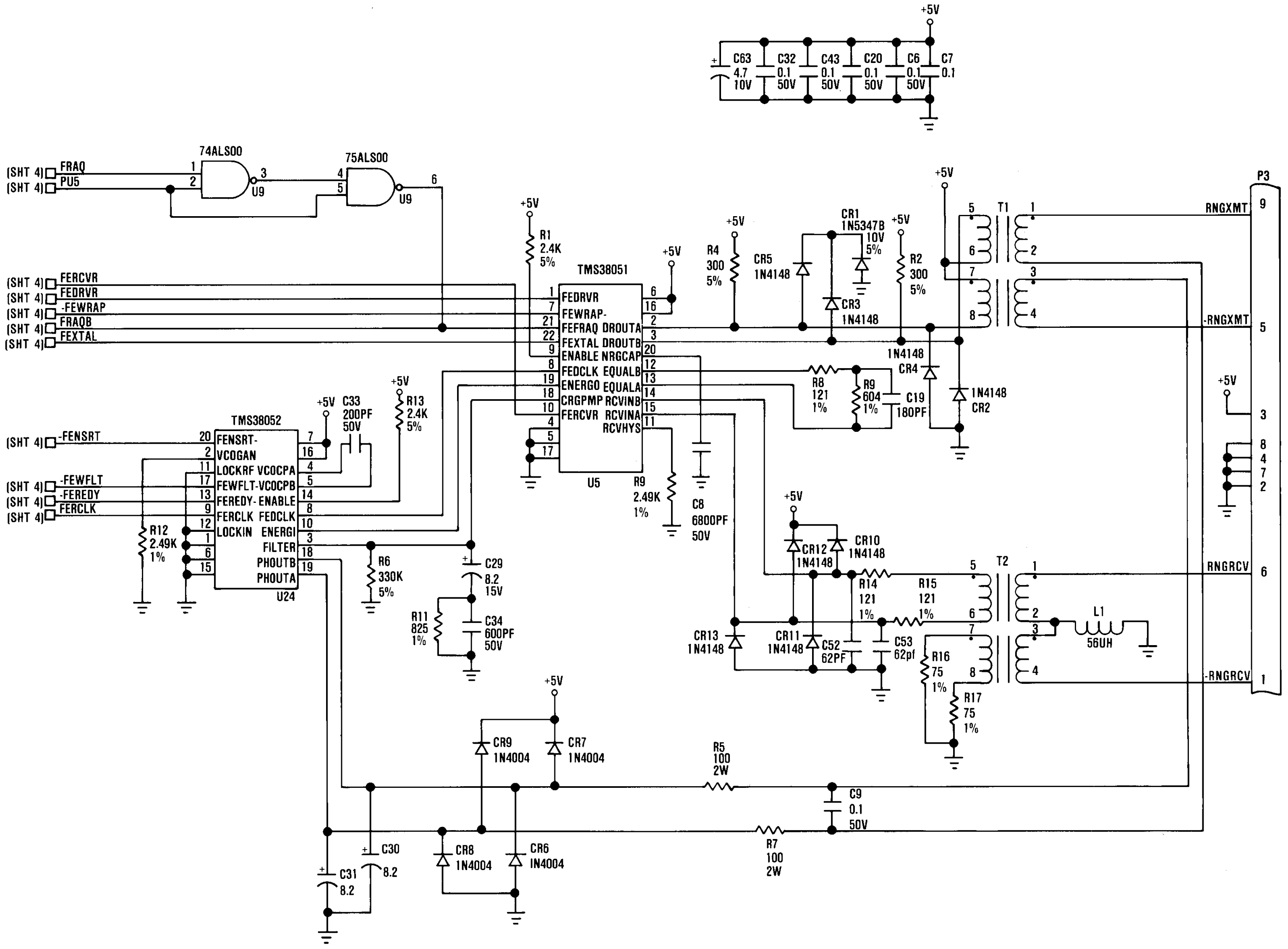


Figure 2-4A. Token-Ring Adapter Logic Diagram (card assembly 56X2295 or 69X8139) (Sheet 6)

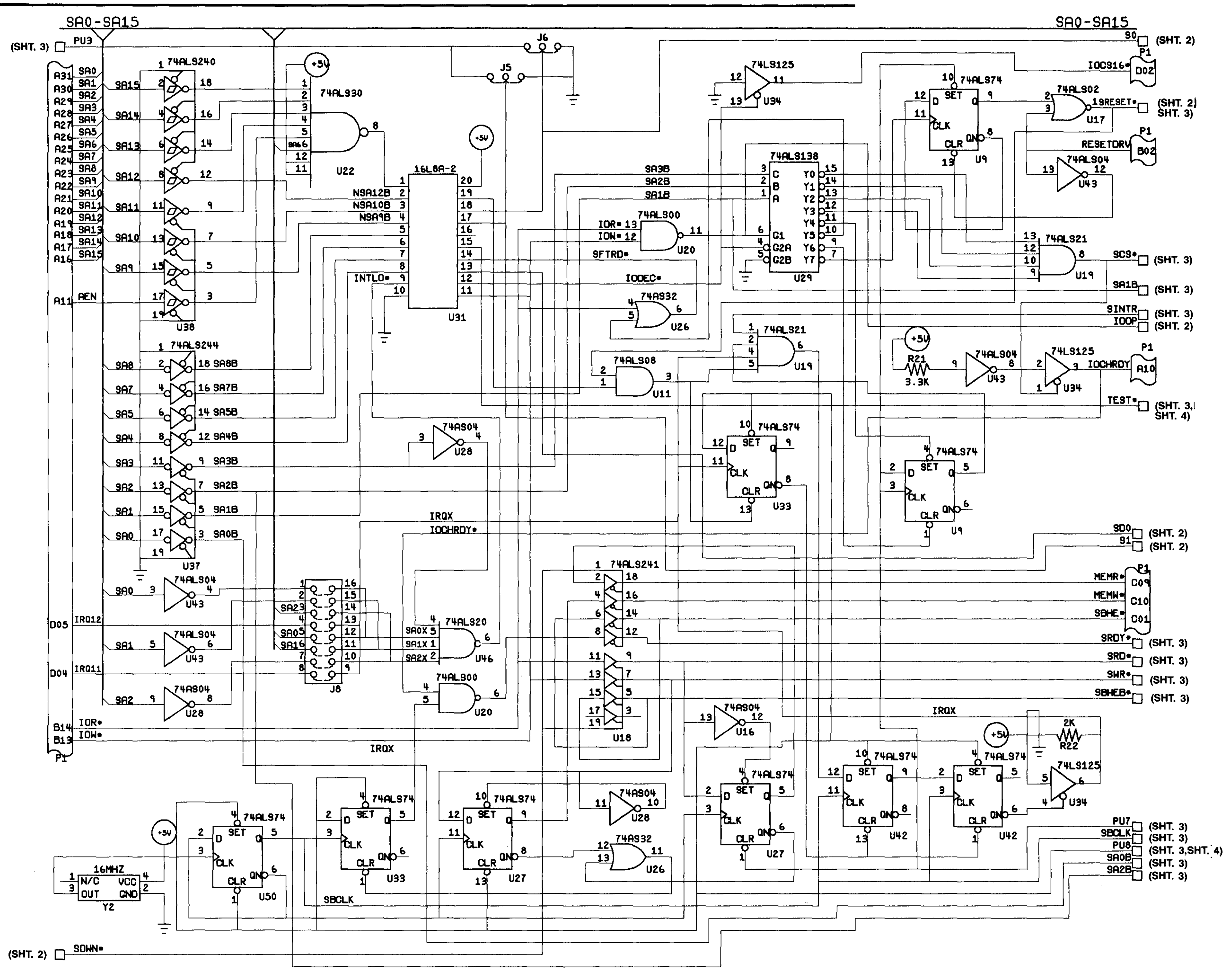


Figure 2-4B. Token-Ring Adapter Logic Diagram (card assemblies other than 56X2295 and 69X8139) (Sheet 1)

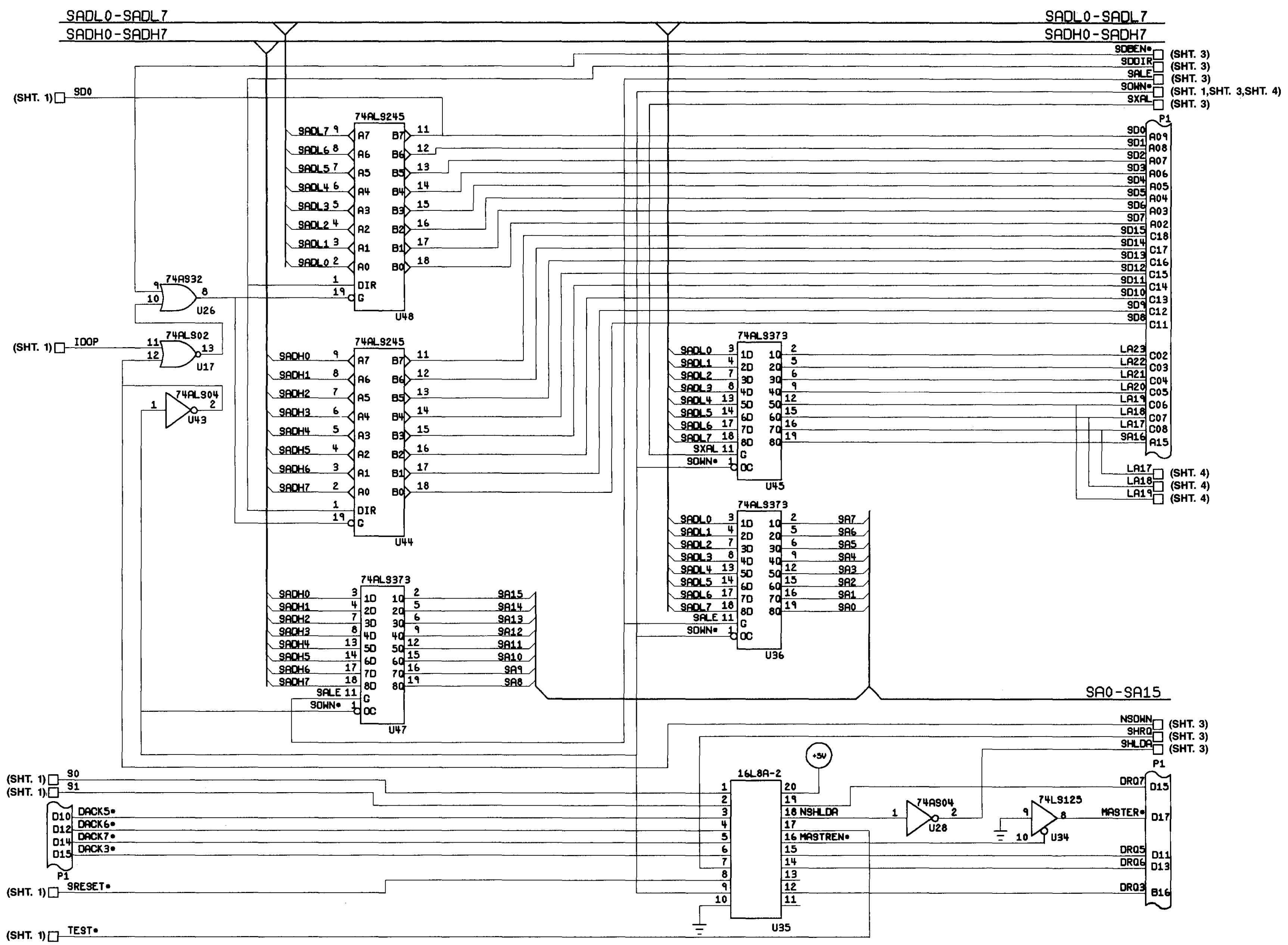


Figure 2-4B. Token-Ring Adapter Logic Diagram (card assemblies other than 56X2295 and 69X8139) (Sheet 2)

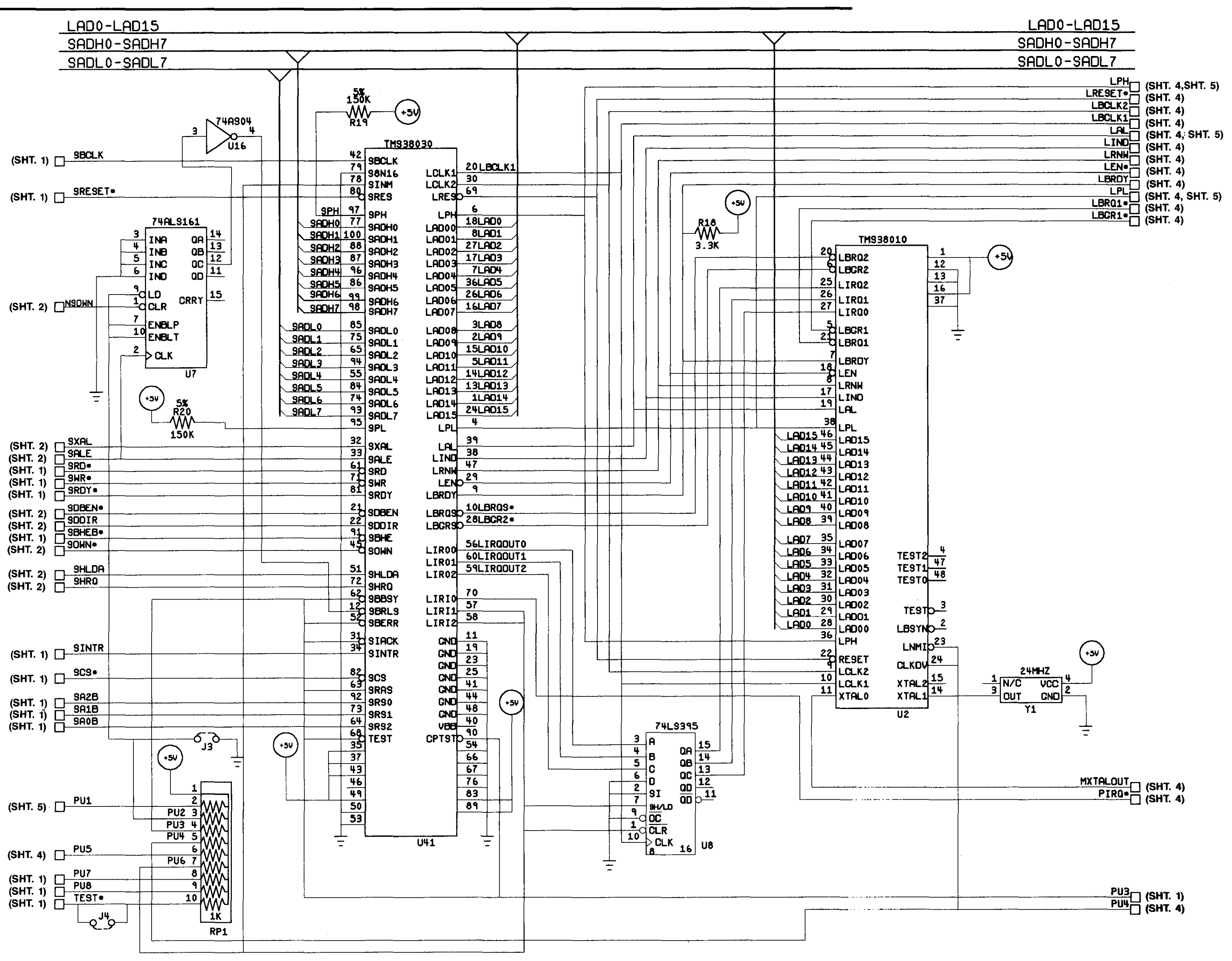


Figure 2-4B. Token-Ring Adapter Logic Diagram (card assemblies other than 56X2295 and 69X8139) (Sheet 3)

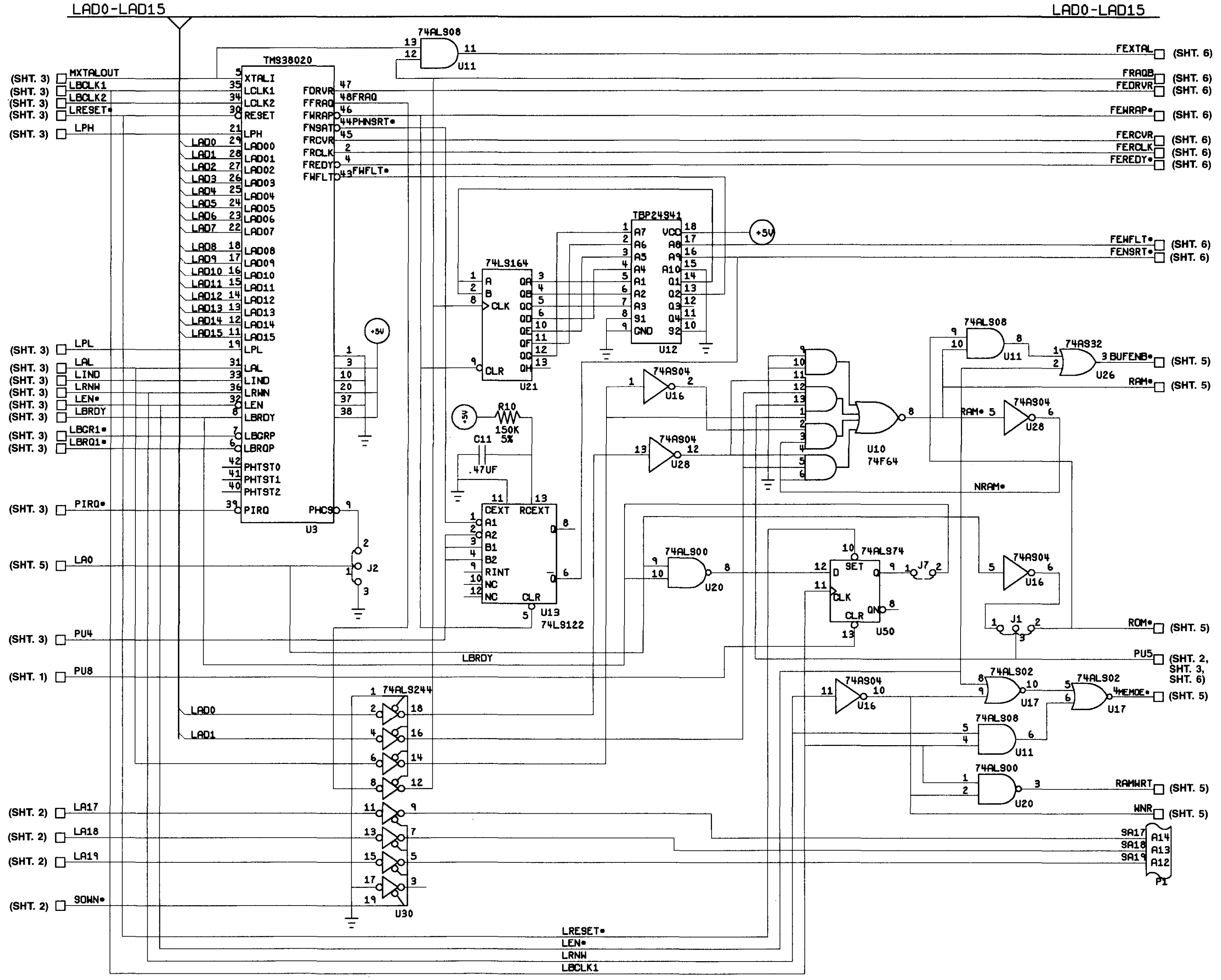


Figure 2-4B. Token-Ring Adapter Logic Diagram (card assemblies other than 56X2295 and 69X8139) (Sheet 4)

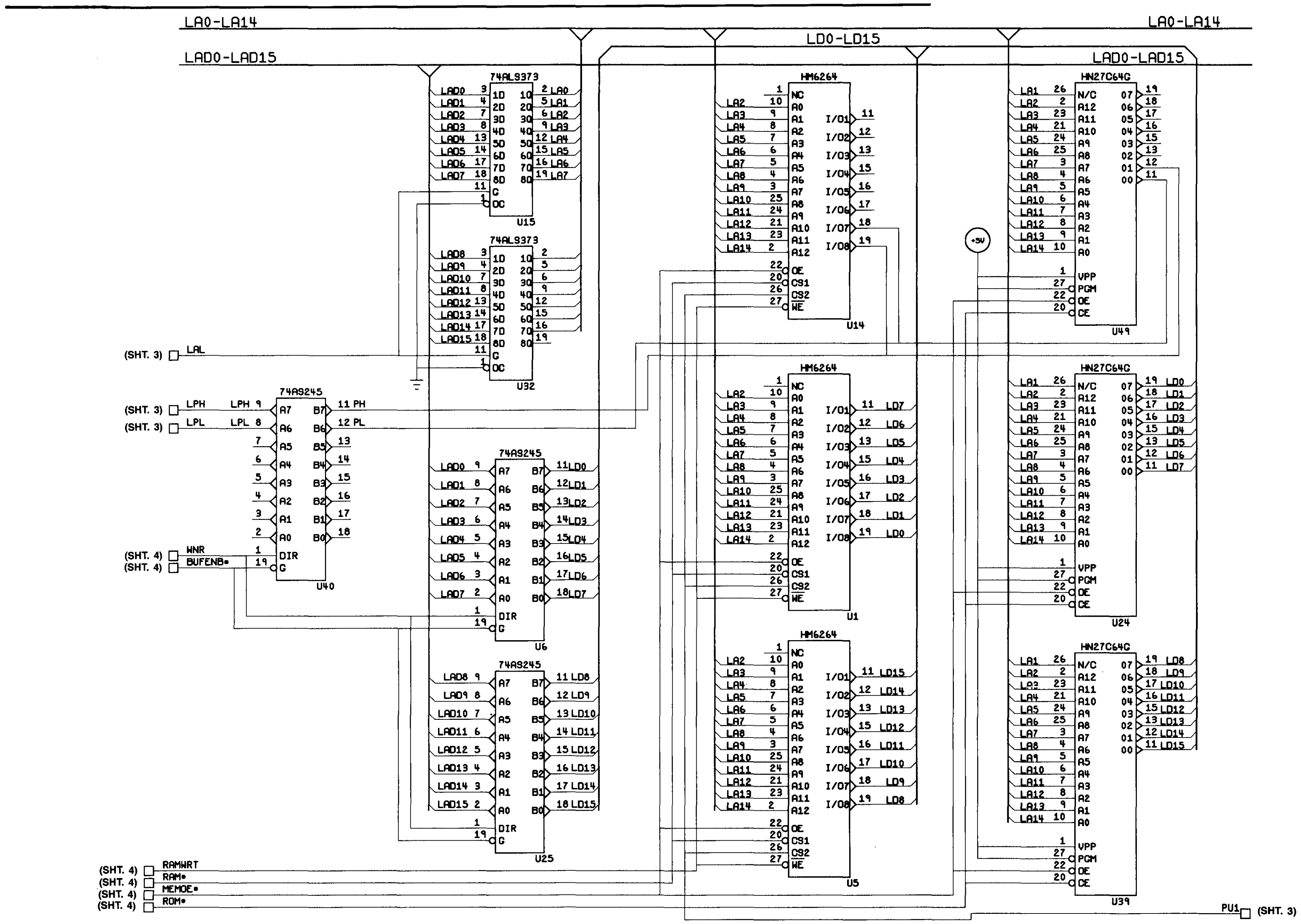


Figure 2-4B. Token-Ring Adapter Logic Diagram (card assemblies other than 56X2295 and 69X8139) (Sheet 5)

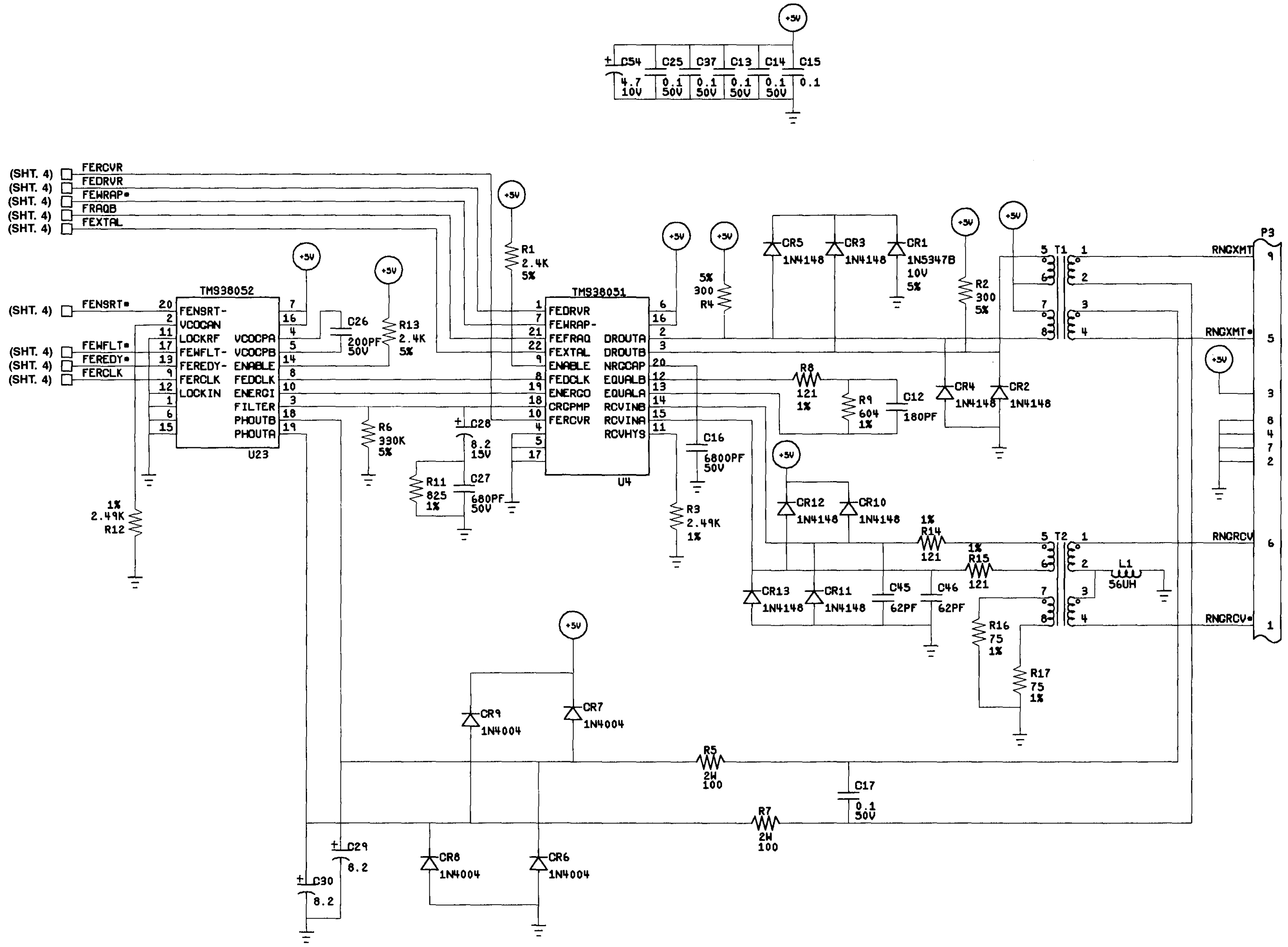


Figure 2-4B. Token-Ring Adapter Logic Diagram (card assemblies other than 56X2295 and 69X8139) (Sheet 6)

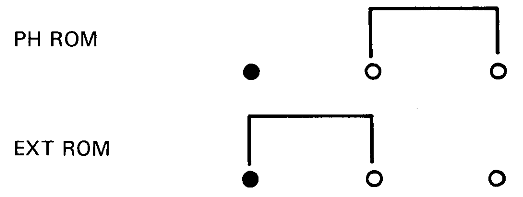
Physical Description

This data defines the physical aspects of the Adapter card including the jumpers and the component placement.

Jumper Descriptions

The Adapter Card features five selectable jumpers which allow the user to reconfigure the card. The following diagrams identify the jumper location and give a brief description of their function. The dot (•) adjacent to each jumper indicates pin 1 position. The default positions are listed in Tables 2-4A and 2-4B.

JUMPERS J1 and J2 – ROM Select 1 and 2



● = pin 1

JUMPER J3 – Primary/Secondary Address and DMA Level Select

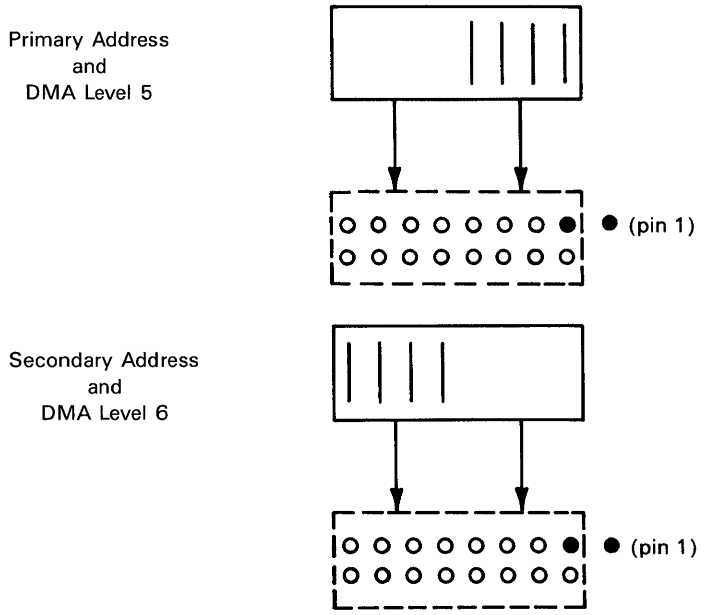


Figure 2-5A. Jumper Diagrams (card assembly 56X2295 or 69X8139) (Sheet 1)

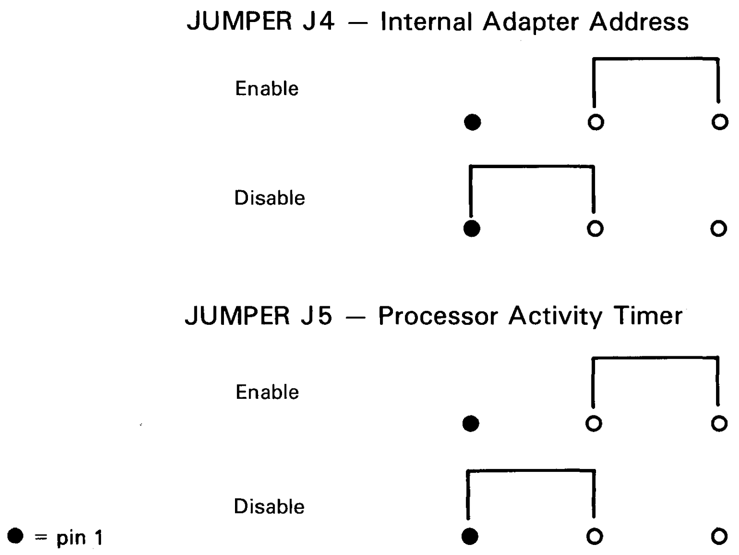

JUMPER J4 — Internal Adapter Address
Enable
Disable
JUMPER J5 — Processor Activity Timer
Enable
Disable
● = pin 1

Figure 2-5A. Jumper Diagrams (card assembly 56X2295 or 69X8139) (Sheet 2)

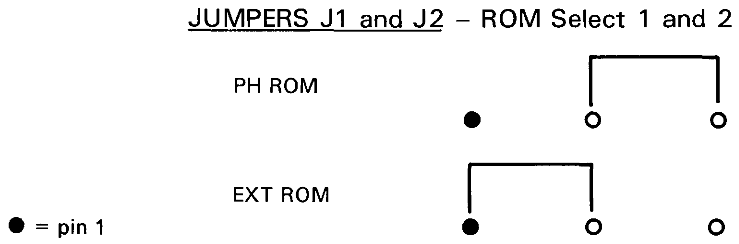
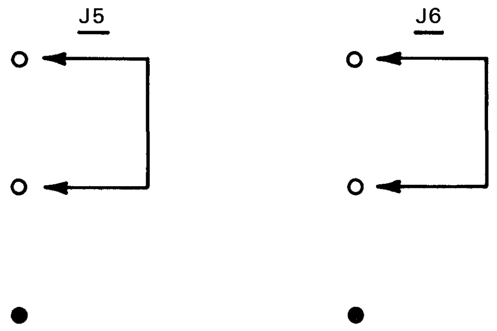


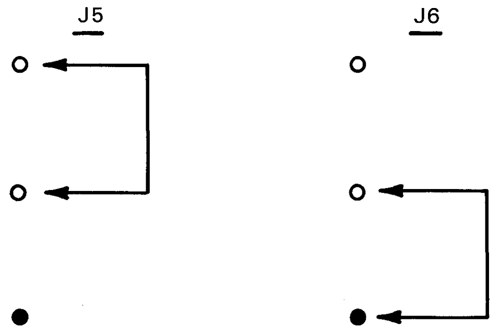
Figure 2-5B. Jumper Diagrams (For card assemblies other than 56X2295 and 69X8139) (Sheet 1)

JUMPERS J5 and J6 – I/O Address Space and DMA Level Select

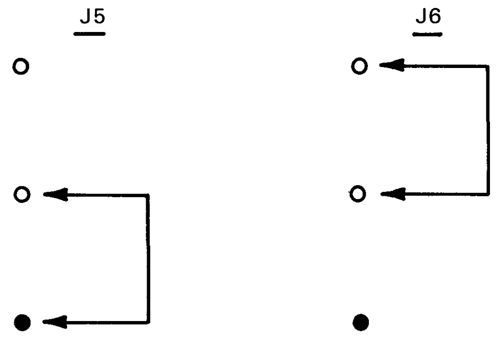
I/O Address Group 1
and
DMA Level 5



I/O Address Group 2
and
DMA Level 6



I/O Address Group 3
and
DMA Level 7

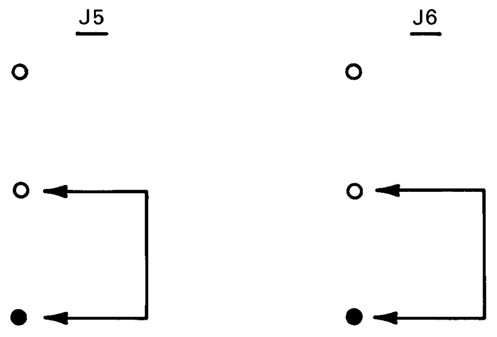


● = pin 1

Figure 2-5B. Jumper Diagrams (For card assemblies other than 56X2295 and 69X8139) (Sheet 2)

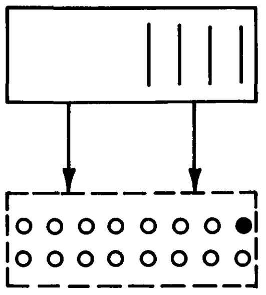
JUMPERS J5 and J6 – I/O Address Space and DMA Level Select (Con't)

I/O Address Group 4
and
DMA Level 3

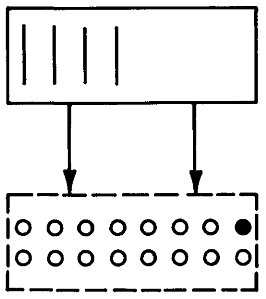


JUMPER J8 – Interrupt Level/Interrupt Enable Select

Interrupt Level 12
and
Interrupt Enable '06F4'



Interrupt Level 11
and
Interrupt Enable '06F3'



● = pin 1

Figure 2-5B. Jumper Diagrams (For card assemblies other than 56X2295 and 69X8139) (Sheet 3)

Jumper Configuration

Table 2-4A. Default Jumper Configuration (card assemblies 56X2295 and 69X8139)

Jumper No.	Default Position
J1	Protocol Handler ROM
J2	Protocol Handler ROM
J3	Primary Address Space
J4	Internal Adapter Address
J5	Processor Activity Timer Included

Table 2-4B. Default Jumper Configuration (other card assemblies)

Jumper No.	Default Position
J1	Protocol Handler ROM
J2	Protocol Handler ROM
J5	I/O Address Group 1 and DMA Level 5
J6	I/O Address Group 1 and DMA Level 5
J8	Interrupt Level 12

Card Layout

The physical component layout of the Adapter Card is shown in Figures 2-6A and 2-6B. In no case will any components be included on the card that will not allow side-by-side mounting of this card with another card in a ¾-inch card to card spacing.

The on-card connector for the Ring-in and Ring-out signals is a 9-pin D-shell connector. The shield on the Token-Ring Network wire media coming from the wall connector is grounded to the product frame through the tailgate. The DC ground of the Adapter is not connected to the chassis ground. Refer to table 2-3 for cable connections.

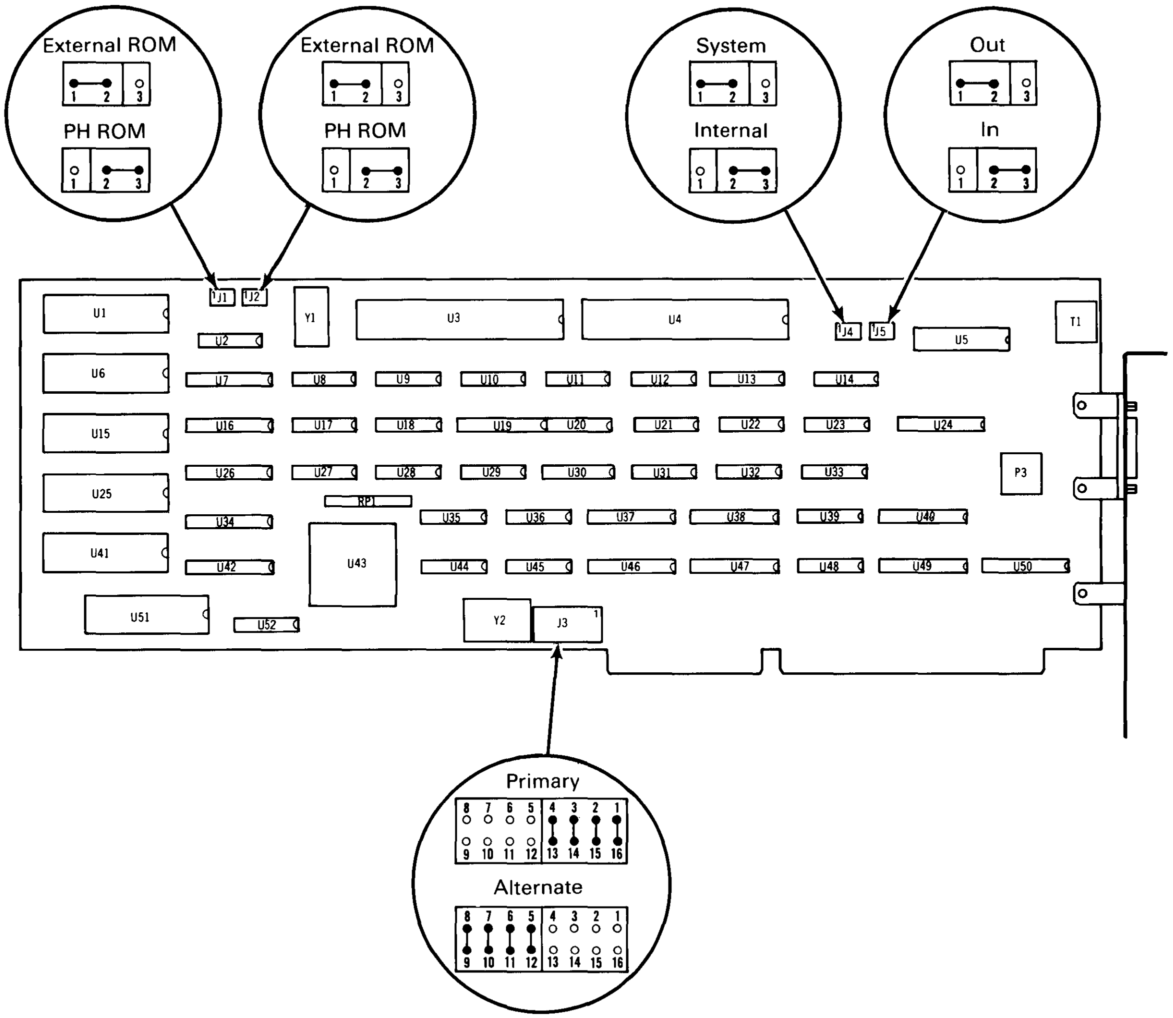


Figure 2-6A. Token-Ring Adapter Card (card assemblies 56X2295 and 69X8139)

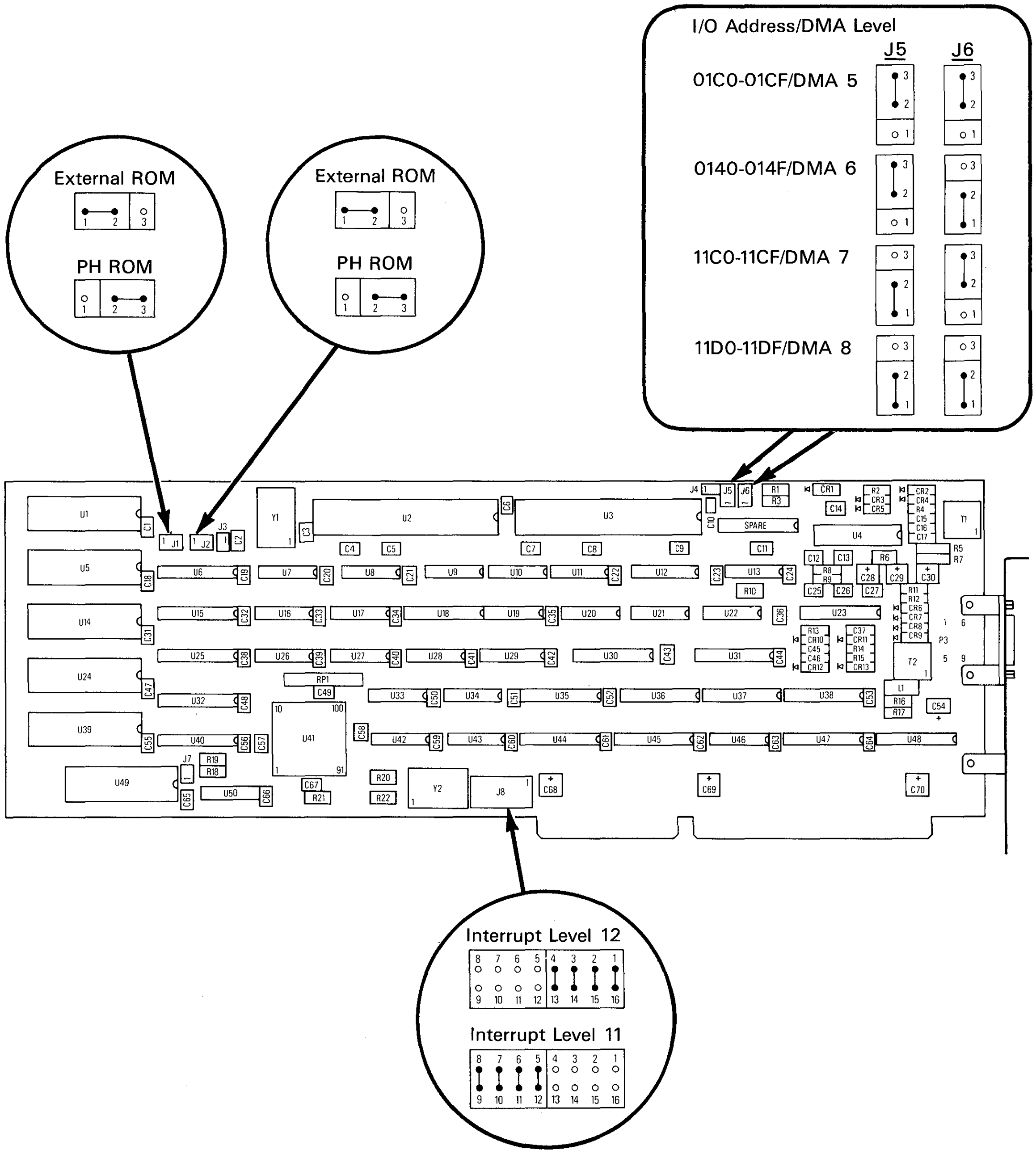


Figure 2-6B. Token-Ring Adapter Card (card assemblies other than 56X2295 or 69X8139)

Chapter 3. Software Interface

CONTENTS

Software Interface

This chapter describes the software interface between the IBM RT Personal Computer (hereafter referred to as "the attached system") and the IBM Token-Ring Network RT PC Adapter (hereafter referred to as Adapter). This software interface controls the operation of the Adapter to effect data transfer to and from the network.

The Adapter is controlled through direct access to four registers and a DMA channel contained within the System Interface (SIF).

The System Interface registers are used to initialize the Adapter, read the cause of interrupts posted to the attached system, and post interrupts to the Adapter to initiate DMA transfers to and from the system memory.

The DMA channel is used to pass commands, parameters, and frames to the Adapter and to receive completion codes and frames from the Adapter. In addition, all data movement to and from the ring is via DMA only. The DMA channel is initialized during Adapter Initialization which is described later in this chapter.

Summary of System Buffer Requirements

The integration of the Adapter into an attached system requires that several system memory buffers be allocated and reserved for Adapter use. The following list specifies the necessary system memory allocation. For more detailed information, refer to "The Command and Status Block" and "Adapter Commands" in this chapter.

SYSTEM COMMAND BLOCK	The System Command Block (SCB) is a six byte buffer which is used to hold the command to be executed by the Adapter and a 24-bit address pointer to a parameter block or buffer.
SYSTEM STATUS BLOCK	The System Status Block (SSB) is an eight byte buffer which is used to hold status codes returned upon completion of Adapter commands.
COMMAND PARAMETER LISTS	Certain commands (like the **OPEN** command) require that a block of memory be designated as a parameter block. Once the command has completed execution, this buffer allocation may be released for other uses.
RECEIVE LIST	The **RECEIVE** command requires that Receive List(s) be allocated within system memory. The memory allocation size is dependent upon the size and number of lists used in the application. The size of a receive list may be selected upon Adapter initialization to be either 14, 20, or 26 bytes in length. The number of lists is application dependent. A discussion on Receive Lists is provided later. Refer to "Receive Commands" in this chapter.
TRANSMIT LIST	The **TRANSMIT** command requires that Transmit List(s) be allocated within system memory. The memory allocation size is dependent upon the size and number of lists used in the application. The size of a Transmit List may be selected upon Adapter initialization to be either 14, 20, or 26 bytes in length. The number of lists is application dependent.
PRODUCT ID BLOCK	The **OPEN** command requires a pointer to an 18 byte Product ID block as part of the open parameter list. The system software designer should reserve 18 bytes of system memory for this function. After completing **OPEN**, the reserved 18 bytes may be released for other uses. Additional information on Product IDs may be found in the *IBM Token-Ring Architecture Reference.*

Register Descriptions

The Adapter contains eight registers which are used for Direct I/O (DIO) reads and writes. (Refer to the table below.) This chapter describes in detail the functions these registers perform.

I/O Address Space Assignments

Group 1	Group 2	Group 3*	Group 4*	
01C0	0140	11C0	11D0	SIF Data
01C2	0142	11C2	11D2	SIF Data INC
01C4	0144	11C4	11D4	SIF Address
01C6	0146	11C6	11D6	Command/Status
01C8	0148	11C8	11D8	Enable Interrupts
01CC	014C	11CC	11DC	Disable Interrupts
01CE	014E	11CE	11DE	Adapter Reset

* Groups 3 and 4 are not supported on card assembly 56X2295 or 69X8139

Command/Status Register

The Command/Status Register is used to post interrupts to the Adapter as well as to read interrupt status information from the Adapter. The function performed by each bit of the CMD/STS Register is dependent upon whether a bit is read or written by the attached system. It is important to note that bits 0-7 can be set to one but not reset to zero by the attached system. These bits, when set to one by the attached system, can only be cleared by the Adapter. Likewise, bit 8 can be reset by zero by the attached system but can only be set by the Adapter. Bits 9-15 can be read only by the attached system. These bits are set or reset by the Adapter.

Writing to the CMD/STS Register

A direct I/O (DIO) write to the CMD/STS Register will transfer a 16-bit word which is used to post interrupts to the Adapter as well as reset an Adapter system interrupt.

Figure 3-1 shows the bit assignments of the CMD/STS Register when written by the attached system. Table 3-1 defines the functions of each bit.

(MSB) Bit 0	Interrupt Adapter
1	Adapter Reset
2	SSB Clear
3	Execute
4	SCB Request
5	Receive Continue
6	Receive Valid
7	Transmit Valid
8	Reset System Interrupt
9	X
10	X
11	X
12	X
13	X
14	X
15 (LSB)	X

'X' denotes don't care

Figure 3-1. CMD/STS Register Write Bit Assignments

Table 3-1. CMD/STS Register Write Bit Functions

BIT 0	INTERRUPT ADAPTER. Bit 0, when set to one, will cause an internal Adapter interrupt. This bit when set to zero has no effect. This bit will be cleared by the Adapter after the Adapter responds to the interrupt. The purpose of the interrupt is defined by the SSB CLEAR, EXECUTE, SCB REQUEST, RECEIVE CONTINUE, RECEIVE VALID, and TRANSMIT VALID bits described below.
BIT 1	ADAPTER RESET. Setting bit 1 to one forces an Adapter reset if bit 0 and bits 2-7 (INTERRUPT ADAPTER, SSB CLEAR, EXECUTE, SCB REQUEST, RECEIVE CONTINUE, RECEIVE VALID, and TRANSMIT VALID) are also set to one. Following an Adapter Reset, the initialization procedure outlined in this chapter should be followed. This reset function is a software command and certain conditions of hardware failure may prevent its execution.
BIT 2	SSB CLEAR. Setting this bit to one activates this interrupt request. This interrupt request is used by the system to notify the Adapter that the System Status Block (SSB) is available for the Adapter to post additional status information.
BIT 3	EXECUTE. Setting this bit to one activates this interrupt. This interrupt is used to initiate an Adapter command contained in the System Command Block (SCB). This block will be DMA read and executed by the Adapter
BIT 4	SCB REQUEST. Setting this bit to one activates this interrupt. This interrupt is used to cause the Adapter to interrupt the attached system when the SCB is available for another command. The Adapter will return the SCB CLEAR interrupt code.
BIT 5	RECEIVE CONTINUE. Setting bit 5 to one activates this interrupt request. This interrupt request signals the Adapter that buffers have been added to the Receive List Chain in the attached system's memory.
BIT 6	RECEIVE VALID. Setting this bit to one activates this interrupt request. This interrupt request signals the Adapter that the condition causing List Processing suspension during receive has been cleared.
BIT 7	TRANSMIT VALID. Setting bit 7 to one activates this interrupt request. This interrupt request signals the adapter that the condition causing List Processing suspension during transmit has been cleared.
BIT 8	RESET SYSTEM INTERRUPT. Writing a zero to bit 8 will reset the Adapter-to-attached system interrupt (i.e. clearing the SINTR line). Writing a one to this bit will inhibit the reset of the system interrupt. SSB CLEAR and INTERRUPT ADAPTER should always be set when this bit position is cleared.
BIT 9-15	RESERVED. These bits are ignored.

Reading from the CMD/STS Register

A direct I/O (DIO) read of the CMD/STS Register will transfer a 16-bit word, which is used to examine status of the Adapter.

Figure 3-2 shows the bit assignments of the CMD/STS Register when read by the attached system. Table 3-2 defines the functions of each bit.

(MSB) Bit 0	Interrupt Adapter
1	Adapter Reset
2	SSB Clear
3	Execute
4	SCB Request
5	Receive Continue
6	Receive Valid
7	Transmit Valid
8	Interrupt System
9	Initialize
10	Test
11	Error
12	Interrupt Code 0 / Error Code 0
13	Interrupt Code 1 / Error Code 1
14	Interrupt Code 2 / Error Code 2
15 (LSB)	Error Code 3

Note: Bits 12 through 15 are used to report bring-up diagnostic and initialization errors. The bring-up diagnostic error codes are shown in Table 3-3 and the initialization error codes are shown in Table 3-5.

Figure 3-2. CMD/STS Register Read Bit Assignments

Table 3-2. CMD/STS Register Read Bit Functions

BIT 0	INTERRUPT ADAPTER. Bit 0 reflects the current state of the system-to-Adapter interrupt bit.
BIT 1	ADAPTER RESET. Bit 1 reflects the current state of the ADAPTER RESET interrupt request bit.
BIT 2	SSB CLEAR. Bit 2 reflects the current state of the SSB CLEAR interrupt request bit.
BIT 3	EXECUTE. Bit 3 reflects the current state of the EXECUTE interrupt request bit.
BIT 4	SCB REQUEST. Bit 4 reflects the current state of the SCB REQUEST interrupt request bit.
BIT 5	RECEIVE CONTINUE. Bit 5 reflects the current state of the RECEIVE CONTINUE interrupt request bit.
BIT 6	RECEIVE VALID. Bit 6 reflects the current state of the RECEIVE VALID interrupt request bit.
BIT 7	TRANSMIT VALID. Bit 7 reflects the current state of the TRANSMIT VALID interrupt request bit.
BIT 8	INTERRUPT SYSTEM. Bit 8 is set to one if the Adapter-to-attached system interrupt is valid. In systems not implementing hardware interrupt control, this bit may be polled under software control. The Adapter cannot reset this bit to zero. This must be done by the attached system writing a zero to this bit location.
BIT 9	INITIALIZE. Bit 9 is set to one when the bring-up diagnostics have completed and the Adapter is ready to start the initialization process. This bit is cleared to zero when the initialization process is complete.
BIT 10	TEST. Bit 10 is set to one by the bring-up diagnostics following an Adapter reset. This bit is cleared when INITIALIZE (bit 9) is initially set.
BIT 11	ERROR. Bit 11 is set if the bring-up diagnostics detect an error or if there is an error during the initialization process. The error condition is specified in bits 12 through 15.

Table 3-2. CMD/STS Register Read Bit Functions (Continued)

BITS 12-14	INTERRUPT CODE. Bits 12-14 define the Adapter-to-attached system interrupt reason code. The lower, the code value numerically, the higher the interrupt priority. The 3-bit interrupt code is shown below:

000	ADAPTER CHECK. This interrupt code is used when the Adapter has encountered an unrecoverable hardware or software error.
001	Reserved.
010	RING STATUS. This interrupt code will be used if the SSB is updated with Ring Status.
011	SCB CLEAR. This interrupt code will be used following a SCB REQUEST interrupt when the SCB is clear.
100	COMMAND STATUS. This interrupt code will be used when the SSB is updated with command status for commands other than **TRANSMIT** and **RECEIVE**. This includes COMMAND REJECT STATUS.
101	RECEIVE STATUS. This interrupt code will be used if the SSB is updated with RECEIVE COMMAND STATUS.
110	TRANSMIT STATUS. This interrupt code will be used if the SSB is updated with TRANSMIT COMMAND STATUS.

Bits 12-15 are also used to indicate the error code resulting from the execution of bring-up diagnostics or the initialization process. The bring-up diagnostic codes are discussed in "Bring-up Diagnostics Verification" in this chapter, and the initialization error codes are discussed in "Adapter Initialization" in this chapter.

Note: Bits 9-15 are only set or reset by the Adapter. The attached system cannot set or reset these bits; these bits can only be read by the attached system.

SIF Address Register

The Address register contains the address pointer for internal Adapter RAM accesses via the Data or Data/Auto-increment registers. All 16 bits can be read, although only bits 5-14 can be actually set/reset by the attached system. This allows the attached system to access a 2k byte block of the Adapter's internal RAM. The actual starting location of RAM which is read is a function of how the Adapter sets bits 0-4. During normal operation of the Adapter, bits 0-4 will be set to 00001. If an Adapter Check Interrupt occurs, these bits will be set to 00000. Bit 15 is controlled by the Adapter as all data transfers on the Adapter bus are by 16-bit words only.

SIF Data Register

The Data Register is the "port" into the Adapter's internal RAM in which data may be read or written. The internal RAM locations read or written through this register are pointed to by the address contained in the Address Register. This capability allows initialization parameters to be passed to the Adapter for initialization.

After initialization, only the address range X'0800' through X'0FFF' may be read through the Data Register. Write access to the internal Adapter Bus is denied.

SIF Data Register with Auto-increment [SIF Data INC]

This register is identical to the Data Register except that the address contained in the Address Register is automatically incremented following a read or write to this register. This allows Adapter RAM to be read or written to sequentially without re-writing the Address Register pointer between each access. If the Address register attempts to increment past X'0FFF', the address will reset to X'0800'.

Enable Interrupt

The Adapter posts a System Interrupt only if the Interrupt has been enabled. The Interrupt is enabled by executing a DIO read or write to the addresses shown above.

Disable Interrupt

The Adapter may be prevented from posting a System Interrupt by disabling the Interrupt. The Interrupt is disabled by executing a DIO read or write to the addresses shown above.

Adapter Reset

The Adapter may be hard reset by the attached system driving RESET DRV or soft reset by executing back-to-back DIO reads or writes to the addresses shown.The first DIO asserts the internal reset and the second deasserts the reset. A reset puts the Adapter into a known state by resetting the internal Adapter Bus. A reset acts to clear any detected interrupt and tri-states the DMA Request lines. Note that since a tri-stated DRQ line is indeterminate, it may be errantly detected by the System DMA controller. The problem can be averted by disabling the DMA controller before a soft reset occurs. A second programming note when using the soft reset feature is that back-to-back DIO reads or writes to the Adapter Reset address may not provide the minimum reset time required by the Adapter. It is recommended that the software provide a wait time of four DIO cycles between the first and second Reset DIO.

Interrupt Level Enable [ILE]

The Interrupt of the Adapter is a shareable interrupt in that a pulse on the IRQxx line effectively disables the interrupt of the adapter until a write to the ILE address clears the detected interrupt. A system reset, whether hard or soft, also clears any latched interrupts.

The ILE address for interrupt level 12 is X'06F4' and for interrupt level 11* is X'06F3'.

* Interrupt level 11 is not supported on card assemblies 56X2295 and 69X8139.

Bring-up Diagnostics

The Adapter executes a stand-alone diagnostic routine upon one of three conditions:

1. Host System asserts RESET DRV
2. Host System executing a DIO read/write to the Adapter Reset.
3. Writing a one to bits 0-7 (X'FF00') of the CMD/STS Register. (All other bits are zero.)

Bring-up Diagnostics Verification

Before the Adapter can be initialized for proper operation, the attached system must verify that the bring-up diagnostics terminated normally. To do this the following procedure should be followed:

1. After either application of the RESET DRV signal, a DIO to the Adapter Reset register, or writing a software reset (X'FF00') to the CMD/STS Register, the attached system should wait 3 seconds and then read the CMD/STS Register.

   a. If the INITIALIZE bit is set to one, and the TEST bit is zero, and the ERROR bit is zero, then the INTERRUPT CODE bits (12-15) should also be zero. This indicates that the bring-up diagnostics completed successfully and the Adapter may now be initialized.

   b. If the TEST and the ERROR bits are set to one, the diagnostics have detected an unrecoverable hardware error. The bring-up error code may be read from bits 12-15. Table 3-3 lists the definitions of these error codes.

   c. If neither of the above conditions occur within three seconds of reset, there is a hardware error preventing completion of the diagnostic routines. The attached system should reset and re-try this procedure three times. If still unsuccessful, an unrecoverable hardware error has occurred and the Adapter should be checked.

Table 3-3. Bring-up Diagnostics Error Codes

Error Code Bits				Error Condition
12	13	14	15	
0	0	0	0	Initial Test Error
0	0	0	1	Adapter ROM CRC Error
0	0	1	0	Adapter RAM Error
0	0	1	1	Instruction Test Error
0	1	0	0	Context/Interrupt Test Error
0	1	0	1	Protocol Handler Hardware Error
0	1	1	0	System Interface Register Error

Following verification of the bring-up diagnostics, the attached system software may now continue with Adapter initialization.

Adapter Initialization

After verification that the Adapter's bring-up diagnostics completed normally, the system software must initialize the Adapter. This initialization involves the transfer of parameters to the Adapter using the DIO interface. These parameters specify:

- The address in the system memory of the System Command Block (SCB) and System Status Block (SSB)
- Interrupt control parameters.

Before the completion of the initialization process, the Adapter initiates a test of the DMA interface. These tests include:

- DMA writes to both the System Command Block and System Status Block
- DMA reads from both the System Command Block and System Status Block to compare to expected results.

These tests do not require any attached system software to execute. However, in the event these tests fail, the Adapter will return an error in the CMD/STS Register.

The Initialization Block

The Initialization Block is 22 bytes (11 16-bit words) in length and the entire block must be transferred to the Adapter. Figure 3-3 defines the 11 words of this block.

	High Byte	Low Byte	
Word 0	Initialization Options		
1	Command	Transmit	
2	Receive	Ring	Interrupt Vectors
3	SCB Clear	Adapter Check	
4	Receive Burst Size		
5	Transmit Burst Size		
6	DMA Abort Thresholds		
7	SCB Address (high)		
8	SCB Address (low)		
9	SSB Address (high)		
10	SSB Address (low)		

Figure 3-3. Parameter Initialization Block

The following information describe the various fields of the Initialization Block and the corresponding bit functions within each field.

Initialization Options

This 16-bit field is used to specify the desired initialization options. The bit assignments of the 16-bit Initialization Options field is shown in Figure 3-4.

(MSB) Bit 0	1
1	Parity Enable Bit 1
2	Parity Enable Bit 2
3	Burst SCB/SSB
4	Burst List
5	Burst List Status
6	Burst Receive Data
7	Burst Transmit Data
8	0
9	0
10	0
11	0
12	0
13	0
14	0
15 (LSB)	0

Figure 3-4. Initialization Options Bit Assignments

The function of each of these bits is described in Table 3-4.

Table 3-4. Initialization Options Field Bit Functions

BIT 0	RESERVED. This bit must be set to one.
BITS 1-2	PARITY ENABLE. These bits should be set to one if the system bus provides odd parity on its data. If parity checking is not desired, these bits should be set to zero. If enabled, parity checking is performed on both DIO and DMA transfers between the Adapter and attached system.
BIT 3	BURST SCB/SSB. If this bit is set to one, the Adapter will transfer the SCB from the system and the SSB to the system in DMA burst mode. The burst size will be six bytes for the SCB read, two bytes for SCB clear, and eight bytes for SSB write. If this bit is set to zero, then these transfers will occur in cycle steal mode. (See Note)
BIT 4	BURST LIST. If this bit is set to one, the Adapter will transfer transmit and receive lists from the system in DMA Burst Mode. The burst size will be the list length with the maximum burst size being 26 bytes. If this bit is set to zero, then cycle steal mode is selected. (See Note)
BIT 5	BURST LIST STATUS. If this bit is set to one, the Adapter will transfer list status data to the system in DMA Burst Mode. If this bit is set to zero, cycle steal mode will be selected. (See Note)
BIT 6	BURST RECEIVE DATA. If this bit is set to one, the Adapter will transfer receive data to the system in DMA burst mode. The burst size is specified in the RECEIVE BURST SIZE field of the Initialization Block. If this bit is set to zero, cycle steal mode is selected. (See Note)
BIT 7	BURST TRANSMIT DATA. If this bit is set to one, the Adapter will transfer transmit data from the system in DMA burst mode. The burst size is specified in the TRANSMIT BURST SIZE field of the Initialization Block. If this bit is set to zero, cycle steal mode is selected. (See Note)
BITS 8-15	RESERVED. These bits must be set to zero.

Note: If burst sizes are small, better perfromance is attained by using cycle steal mode.

Command Status Vector

This byte contains the interrupt vector that the Adapter places on the attached system bus when the SSB is updated with command status for commands other than **TRANSMIT** or **RECEIVE**. COMMAND REJECT STATUS will also use this vector.

Transmit Command Status Vector

This byte contains the interrupt vector that the Adapter places on the attached system bus when the SSB is updated with TRANSMIT COMMAND STATUS.

Receive Command Status Vector

This byte contains the interrupt vector that the Adapter places on the attached system bus when the SSB is updated with RECEIVE COMMAND STATUS.

Ring Status Vector

This byte contains the interrupt vector that the Adapter places on the attached system bus when the SSB is updated with RING STATUS.

SCB Clear Vector

This byte contains the interrupt vector that the Adapter places on the attached system bus if an SCB CLEAR interrupt is generated.

Adapter Check Vector

This byte contains the interrupt vector that the Adapter places on the attached system bus if an ADAPTER CHECK interrupt is generated.

Receive Burst Size

This 16-bit field contains a count of the maximum number of bytes that the Adapter will DMA during one burst cycle when receive data is written to the attached system memory. If this field is cleared, the Adapter will set the burst size equal to the amount of data to be transferred. This parameter is ignored if the BURST RECEIVE DATA bit of the Initialization Options field is set to zero, indicating cycle steal mode. This parameter must be even.

Transmit Burst Size

This 16-bit field contains a count of the maximum number of bytes that the Adapter will DMA during one burst cycle when transmit data is read from the attached system memory. If this field is cleared, the Adapter will set the burst size equal to the amount of data to be transferred. This parameter must be even since only two byte transfers occur. This parameter is not ignored if the BURST TRANSMIT DATA bit of the Initialization Options field is set to zero, indicating cycle steal mode. Even in cycle steal mode, TRANSMIT BURST SIZE is used to limit transmit data bus utilization so that higher priority receive DMA operations can be initiated.

DMA Abort Thresholds

This 16-bit field contains counts for the number of times the Adapter will attempt a DMA operation (read or write) if it is terminated abnormally with a bus error or parity error. The high-order byte (bits 0-7) contains the count for bus errors and the low-order byte (bits 8-15) contains the count for parity errors. The counts specify the total number of times the DMA operation is to be attempted. A count of zero is not permitted.

SCB Address

This 32-bit field contains the 24-bit address of the SCB in attached system memory. This value must be an even address aligned on a word boundary. The high-order byte of this field is ignored.

SSB Address

This 32-bit field contains the 24-bit address of the SSB in attached system memory. This value must be an even address aligned on a word boundary. The high-order byte of this field is ignored.

Writing the Initialization Block

The Initialization Block is passed to the Adapter by following the procedure below:

1. Verify that the bring-up diagnostics completed successfully as previously described.
2. Write the address X'0200' into the SIF Address Register.
3. Begin transfer of the Initialization Block by writing each byte or 16-bit word to the SIF Data Register/Auto-increment. This will cause the block to be written to successive Adapter RAM locations beginning at RAM address X'0A00'.
4. Write the bit pattern X'9080' to the CMD/STS Register. This sets the INTERRUPT ADAPTER, EXECUTE, and RESET SYSTEM bits and clears all others.
5. Continue to read the CMD/STS Register until one of the following occurs:
    a. The INITIALIZE, TEST, and ERROR bits are all zero. This condition indicates that initialization is complete without error. The SCB should contain X'0000C1E2D48B' and the SSB should contain X'FFFFD1D7C5D9C3D4'.
    b. If the ERROR bit is set, the initialization process has failed. The Interrupt Code bits 12-15 will contain the initialization error code. (These error codes are listed in Table 3-5.) The initialization procedure must be restarted from Adapter reset.
    c. If neither of the above conditions occurs within 11 seconds of loading the Parameter Initialization Block, there is a hardware error. Because the DMA Timeout Error is not realized for at least 10 seconds, it is recommended that the attached system wait at least 11 seconds to detect an initialization error code. If no Interrupt Code is posted then the Adapter has a hardware error. It is recommended that the attached system reset the Adapter and re-try the initialization procedure three times. If still unsuccessful, there is an unrecoverable hardware error.

Table 3-5. Adapter Initialization Errors

Error Code Bits 12	13	14	15	Error Condition
0	0	0	1	Invalid Initialization Block. Twenty-two (22) bytes were not passed.
0	0	1	0	Invalid Options. This code is returned if the Initialization Options word has the TRANSFER MODE bit set, the PARITY ENABLE Bits are not equal, or the Reserved bits are not zero.
0	0	1	1	Invalid Receive Burst Count. The Receive Burst count is odd.
0	1	0	0	Invalid Transmit Burst Count. The transmit burst count is odd.
0	1	0	1	Invalid DMA Abort Threshold. The DMA abort thresholds were specified as zero.
0	1	1	0	Invalid SCB. The SCB address was specified as odd.
0	1	1	1	Invalid SSB. The SSB address was specified as odd.
1	0	0	0	DIO Parity. A parity error occurred during a DIO write operation.
1	0	0	1	DMA Timeout. The Adapter timed out (10 seconds) waiting for a test DMA transfer to complete.
1	0	1	0	DMA Parity Error. A parity error occurred during the DMA tests and the operation was tried the number of times specified by the DMA Abort Threshold.
1	0	1	1	DMA Bus Error. The DMA test encountered a bus error and the operation was tried the number of times specified by the DMA Abort Threshold
1	1	0	0	DMA Data Error. Initialize DMA test failed due to a data compare error.
1	1	0	1	Adapter Check. The Adapter encountered an unrecoverable hardware error.

The Command and Status Block

Two fixed-address control blocks must be provided by the system: the System Command Block (SCB) and the System Status Block (SSB). The starting address of both blocks is passed to the Adapter during the initialization process as described in the previous section. Both blocks must be aligned on an even byte boundary.

In general, the attached system issues a command to the Adapter by loading the request in the SCB and interrupting the Adapter. The Adapter will then download the command (and any required parameters) through the System Interface DMA channel. If the SCB REQUEST bit (bit 4) of the CMD/STS Register is set, the Adapter will interrupt the attached system after the command has been downloaded, indicating to the attached system that the SCB is available for additional commands.

When the status of any outstanding command is to be returned, the Adapter will load the SSB via DMA and interrupt (if enabled) the attached system. After the system has read the SSB, the system must notify the Adapter that the SSB is clear and available for additional status posting. This is done by writing a one to the SSB CLEAR bit (bit 2) of the Adapter's CMD/STS Register. All command status is returned in the SSB.

Command Initiation: System Command Block

The System Command Block (SCB) is six bytes in length and the Adapter will always DMA read 6 bytes. The SCB format is shown in Figure 3-5.

SCB Address	+ 0	Command
	+ 2	Address (high)
	+ 4	Address (low)

Figure 3-5. System Command Block Format

The Command field contains the 16-bit command code request to the Adapter. The command set of the Adapter will be discussed in detail later in this chapter.

The 32-bit address field contains a 24-bit address used as a pointer to the command parameters. The high-order byte of this field is read but ignored. Some commands do not have additional parameters and only the 16-bit command code must be written.

The attached system initiates an Adapter command by following the sequence shown below:

1. The attached system must write the command request code into the SCB including the address to the parameter block, if required.

2. The attached system writes to the Adapter's CMD/STS Register and sets the following bits to one:

   a. INTERRUPT ADAPTER bit (bit 0)

   b. SSB CLEAR bit (bit 2)

   c. EXECUTE bit (bit 3).

Note: The RESET SYSTEM INTERRUPT bit (bit 8) must be considered and properly set to prevent loss of interrupts.

This sequence will cause an interrupt internal to the Adapter. The Adapter will fetch, via DMA, the SCB and any required parameters. Once the SCB and any required parameters are downloaded, the Adapter will write a zero to the Command field of the SCB. This indicates to the attached system that the command has been downloaded and another may be issued.

If the SCB REQUEST bit (bit 4) of the Adapter CMD/STS Register is set to one, an interrupt will also be posted to the attached system when the SCB is available for additional commands. If this is the case, the attached system must reset the interrupt by writing a zero to the RESET SYSTEM bit (bit 8) of the CMD/STS Register to clear the Adapter-to-system interrupt and post an ILE to clear the interrupt. The system should also check that the Command Field of the SCB is set to zero when the interrupt is recognized. If the Command Field is zero, then the SCB is available for use. If the Command Field is non-zero, an EXECUTE Interrupt request was issued or the SCB was altered in preparation for an EXECUTE Interrupt request subsequent to the SCB REQUEST. If SCB REQUEST is desired, it is recommended that either the SCB REQUEST be issued coincident with an EXECUTE Interrupt Request, or that the SCB alteration and EXECUTE Interrupt Request be performed only in response to SCB CLEAR.

A maximum of three commands may be executed simultaneously within the Adapter. A fourth command will not begin execution until there are less than three commands executing simultaneously. There may not be more than one transmit or receive command executed at one time. Thus the Adapter may be executing a **TRANSMIT** command, a **RECEIVE** command, and one other command.

Status Reporting: The System Status Block

The System Status Block (SSB) is eight bytes in length. It is the block in which the Adapter returns RING STATUS, COMMAND REJECT STATUS, and status upon completion of Adapter commands. The Adapter will always DMA write the entire eight bytes regardless of the actual length of the returned status. The unspecified status fields should be ignored in this case. The SSB is not used to return status for frame commands. The frame status information can be obtained in the parameter lists associated with **RECEIVE** and **TRANSMIT** commands. The SSB format is defined in Figure 3-6.

SSB Address	+ 0	Command
	+ 2	Status 0
	+ 4	Status 1
	+ 6	Status 2

Figure 3-6. System Status Block Format

The Command field is written to the SSB by the Adapter and is used to identify either RING STATUS, COMMAND REJECT STATUS, or the status of a general command. The Status fields contain the actual status information for the Command field.

Following the DMA operation to write the status information to the SSB, the Adapter will interrupt the attached system to indicate that the SSB contains valid status information. The attached system software should reset the Adapter-to-system interrupt and communicate to the Adapter that the SSB is clear and available for additional status posting. This is done by writing a one to the SSB CLEAR and zero to the RESET SYSTEM INTERRUPT bit (bits 2 and 8 respectively) of the CMD/STS Register. In addition, the attached system must clear the Adapter interrupt by writing an ILE.

RING STATUS Interrupt

The SSB will be loaded with the current ring status and an interrupt posted to the attached system when any of the following error interrupt conditions occur:

1. The Adapter detects a signal loss on the ring.
2. The Adapter is transmitting or receiving beacon frames to/from the ring. This interrupt condition may be disabled by setting bit 1 of the **OPEN** command options to zero.
3. The Adapter transmits a Report Error MAC Frame. This interrupt error condition may be disabled by setting bit 2 of the **OPEN** command options to zero.
4. An open or short circuit fault is detected by the Adapter.
5. The Adapter receives a Remove Ring Station MAC frame.
6. The attached product counter has been incremented from 254 to 255.
7. The Adapter has been opened and is the only station on the ring or becomes the only station on the ring.
8. The Adapter is transmitting or receiving claim Token MAC frames.

Ring Status will not be reported until the completion of the **OPEN** command.

The SSB is loaded with RING STATUS as shown in Figure 3-7. The bit positions of the Ring Status Field are defined in Figure 3-8.

SSB Address	+ 0	X'0001'
	+ 2	Ring Status Field

Figure 3-7. RING STATUS SSB Format

(MSB) Bit 0	Signal Loss
1	Hard Error
2	Soft Error
3	Transmit Beacon
4	Lobe Wire Fault
5	Auto-removal Error
6	-reserved-
7	Remove Received
8	Counter Overflow
9	Single Station
10	Ring Recorvery
11	0
12	0
13	0
14	0
15 (LSB)	0

Figure 3-8. RING STATUS Field Bit Assignments

These bits are described in Table 3-6.

Table 3-6. RING STATUS Field Bit Functions

BIT 0	SIGNAL LOSS. Bit 0, when set to one, indicates that the Adapter has detected a loss of signal on the ring.
BIT 1	HARD ERROR. Bit 1, when set to one, indicates that the Adapter is presently transmitting or receiving beacon frames to or from the ring.
BIT 2	SOFT ERROR. Bit 2, when set to one, indicates that the Adapter has transmitted a Report Error MAC frame.
BIT 3	TRANSMIT BEACON. Bit 3, when set to one, indicates that the Adapter is transmitting beacon frames to the ring.
BIT 4	LOBE WIRE FAULT. Bit 4, when set to one, indicates that the Adapter has detected an open or short circuit in the lobe data path. The Adapter will be closed and at the state following Adapter initialization (waiting for an **SCB** command).
BIT 5	AUTO-REMOVAL ERROR . Bit 5, when set to one, indicates that the Adapter has detected an internal hardware error following the Beacon Auto-removal process and has de-inserted from the ring. The Adapter will be closed and at the state following Adapter initialization (waiting for an **SCB** command).
BIT 6	RESERVED. This bit is undefined.
BIT 7	REMOVE RECEIVED. Bit 7 , when set to one, indicates that the Adapter has received a Remove Ring Station MAC frame request. The Adapter will be closed and at the state following Adapter initialization (waiting for an **SCB** command).
BIT 8	COUNTER OVERFLOW. Bit 8, when set to one, indicates that the Attached Product Counter has been incremented from 254 to 255.
BIT 9	SINGLE STATION. Bit 9, when set to one, indicates that the Adapter has sensed that it is the only station on the ring. This bit will be reset to zero when another station inserts into the ring.
BIT 10	**RING RECOVERY. Bit 10, when set to one, indicates the Adapter is either transmitting or receiving Claim Token MAC frames. This bit is reset to zero when the Adapter receives a Ring Purge MAC frame.**
BITS 11-15	RESERVED. Bits 11-15 will be set to zero.

Some of the RING STATUS Interrupts require a response from the Adapter or the Adapter user. For example, item 4 describes a situation in which an open or short has occurred on the ring lobe media. An appropriate response by the user is to check the ring cable of the workstation for an accidental disconnect from the ring. Item 5 describes the Adapter being forcibly removed from the ring, the user should respond by executing the Adapter Diagnostics to check for proper Adapter operation. If the Adapter successfully completes the Diagnostics, the user may again insert onto the ring. If the user receives a Counter Overflow, the proper action is to do a READ ERROR LOG, which will reset the product counter. Refer to "Read Error Log." in this chapter.

In the event the Adapter receives a Single Station status, the Adapter should ignore the error if it is known to be the first station on the ring or the only station on the ring. If the Adapter receives a Single Station status and there are other stations on the ring, the user should execute the Adapter Diagnostics to ensure proper Adapter operation.

Due to the dynamic nature of the report indications and the possibility that the ring status could change before the system can respond to a previous RING STATUS interrupt, the current ring status could possibly equal the last ring status report.

COMMAND REJECT Status Interrupt

The SSB will be loaded with COMMAND REJECT STATUS if the Command Field or Address Field of the SCB are in error. The format of the SSB in this situation is shown in Figure 3-9. The Command Field of the SSB will be set to X'0002'. The Reject Command will be loaded with the Command Field of the offending SCB. Figure 3-10 defines the bit position within the Reject Reason Field and Table 3-7 describes these bits.

SSB Address	+ 0	X'0002'
	+ 2	Reject Reason
	+ 4	Reject Command

Figure 3-9. COMMAND REJECT SSB Format

(MSB) Bit 0	Illegal Command
1	Address Error
2	Adapter Open
3	Adapter Closed
4	Same Command
5	0
6	0
7	0
8	0
9	0
10	0
11	0
12	0
13	0
14	0
15 (LSB)	0

Figure 3-10. Reject Reason Field Bit Assignments

These bits are described in Table 3-7:

Table 3-7. Reject Reason Field Bit Functions

BIT 0	ILLEGAL COMMAND. Bit 0 is set to one when an illegal command is issued from the SCB.
BIT 1	ADDRESS ERROR. Bit 1 is set to one if the SCB ADDRESS Field is odd (not half-word aligned).
BIT 2	ADAPTER OPEN. Bit 2 is set to one if a command is issued when an Adapter is open and the command is honored only when the Adapter is closed.
BIT 3	ADAPTER CLOSED. Bit 3 is set to one if a command is issued when the Adapter is closed and the command is honored only when the Adapter is open.
BIT 4	SAME COMMAND. Bit 4 is set to one if a command is issued and the same command is already executing.
BITS 5-15	These bits will be set to zero.

ADAPTER CHECK Error Interrupt

The ADAPTER CHECK Interrupt is generated when the Adapter has encountered an unrecoverable hardware or software error. The SSB is not altered when the ADAPTER CHECK Interrupt is generated. The Adapter will be in a closed state waiting for an Adapter reset.

ADAPTER CHECK information may be obtained by writing the 16-bit Address Register with the address X'05E0' and then reading the next consecutive eight bytes through the Data/ Auto-increment register. The Adapter check status format is shown in Figure 3-11. The bit assignments are illustrated in Figure 3-12 and are described in Table 3-8.

Adapter RAM X'05E0'	+ 0	Adapter Check
	+ 2	Parameter 0
	+ 4	Parameter 1
	+ 6	Parameter 2

Figure 3-11. Adapter Check Status Format

(MSB) Bit 0	DIO Parity
1	DMA Read Abort
2	DMA Write Abort
3	Illegal Op Code
4	LB Parity Error
5	EM Parity Error
6	SIF Parity Error
7	PH Parity Error
8	RECV Parity Error
9	XMIT Parity Error
10	Ring Underrun
11	Ring Overrun
12	Invalid Interrupt
13	Invalid Error Interrupt
14	Invalid XOP
15 (LSB)	Program Check

Figure 3-12. Adapter Check Field Bit Assignments

Table 3-8. Adapter Check Bit Definitions

BIT 0	DIO PARITY. Bit 0 is set to one if the Adapter detects bad parity on data passed from the attached system to the Adapter through a direct I/O access. Parameters 0-2 should be ignored.
BIT 1	DMA READ ABORT. Bit 1 is set to one if the Adapter aborts a DMA read operation (from the system). This can be a result of parity errors in excess of the parity abort threshold set during initialization, bus errors in excess of the bus error abort threshold also set during initialization, or if the Adapter times out (10 seconds) waiting for the completion of a DMA bus operation (with or without an error). Parameter 0 will contain the following information: X'0000' Indicates a timeout abort. X'0001' Indicates a parity error abort. X'0002' Indicates a bus error abort. Parameters 1-2 will contain the failing system address. This address can be within plus or minus 6 bytes of the actual failed address.
BIT 2	DMA WRITE ABORT. Bit 2 is set to one if the Adapter aborts a DMA write. The description for DMA READ ABORT (bit 1) applies to this condition also.
BIT 3	ILLEGAL OP CODE. Bit 3 is set to one if the Adapter's Communications Processor detects an illegal operation code in the Adapter's internal program. Parameters 0-2 will contain the Communications Processor registers R13, R14, and R15 respectively.
BIT 4-9	PARITY ERRORS. These bits are set to one if the Adapter detects a bus parity error on the Adapter's internal Adapter Bus. Parameters 0-2 will contain the Communications Processor registers R13, R14, and R15 respectively. The specific bit set to one (bits 4-9) depends upon the source of the error. A description of bits 4-9 and the parity error causing the bits to be set to one, follows: BIT 4 Bit 4 is set to one if the Communications Processor detects the Adapter Bus parity error. BIT 5 Reserved. BIT 6 Bit 6 is set to one if the System Interface detects the Adapter Bus parity error. BIT 7 Bit 7 is set to one if the Protocol Handler detects the Adapter Bus parity error.

Table 3-8. Adapter Check Bit Definitions (Continued)

	BIT 8	Bit 8 is set to one if the parity error occurred when the Adapter was copying a frame from the ring. Parameter 0 will contain the buffer address.
	BIT 9	Bit 9 is set to one if the parity error occurred when the Adapter was transmitting on the ring. Parameter 0 will contain the buffer address.
BIT 10	RING UNDERRUN. Bit 10 is set to one if the Adapter detects an internal DMA underrun when transmitting on the ring. Parameters 0-2 should be ignored.	
BIT 11	RING OVERRUN. Bit 11 is set to one if the Adapter detects an internal DMA overrun when receiving from the ring.	
BIT 12	INVALID INTERRUPT. Bit 12 is set to one if an unrecognized interrupt was generated internal to the Adapter. Parameters 0-2 will contain the Communications Processor registers R13, R14, and R15, respectively.	
BIT 13	INVALID ERROR INTERRUPT. Bit 13 is set to a one if an unrecognized error interrupt was generated. Parameters 0-2 will contain Adapter registers R13, R14, R15.	
BIT 14	INVALID XOP. Bit 14 is set to one if an unrecognized XOP request was generated in the Communications Processor's code. Parameters 0-2 will contain the Communications Processor registers R13, R14, and R15, respectively.	
BIT 15	PROGRAM CHECK. This bit will be set to one if the Adapter's internal software detects a software error. Parameters 0-2 will contain the following:	
	Parameter 0	Abend Code.
	Parameter 1	Location of detected error.
	Parameter 2	Ignored.

Adapter Commands

This information describes the Adapter SCB commands and subsequent error reporting. Table 3-9 lists the available Adapter commands.

Table 3-9. Adapter Command Summary

Function	Command
OPEN	X'0003'
TRANSMIT	X'0004'
TRANSMIT HALT	X'0005'
RECEIVE	X'0006'
CLOSE	X'0007'
SET GROUP ADDRESS	X'0008'
SET FUNCTIONAL ADDRESS	X'0009'
READ ERROR LOG	X'000A'
READ ADAPTER BUFFER	X'000B'

OPEN Command

Before the Adapter can be used for data communications, the attached system must first open the Adapter by issuing an **OPEN** command. The **OPEN** command serves to set the Adapter's various addresses and enables the receipt of frames from the ring. A **RECEIVE** command must be issued immediately after successful **OPEN** completion. The Adapter will suspend processing of all interrupt requests except reset during the **OPEN** process.

The **OPEN** options can be changed only by closing the Adapter via the **CLOSE** command and then re-opening the Adapter with the desired options.

OPEN Command Block

The OPEN SCB is shown in Figure 3-13. The Address Field is a 24-bit address which points to a 32-byte block of the OPEN parameter list. The high order byte of this address is ignored.

SCB Address	+ 0	X'0003'
	+ 2	Address (High)
	+ 4	Address (Low)

Figure 3-13. OPEN Command SCB

Upon completion of the **OPEN** command, the status of the **OPEN** completion is loaded into the SSB address+2.

OPEN Parameter List.

Figure 3-14 defines the **OPEN** parameter list. Table 3-10 describes the functions of the fields within the **OPEN** parameter list.

BYTE	
0	Open Options
2	Node Address (high)
4	Node Address
6	Node Address (low)
8	Group Address (high)
10	Group Address (low)
12	Functional Address (high)
14	Functional Address (low)
16	Receive List Size
18	Transmit List Size
20	Buffer Size
22	Exp. RAM Start Address
24	Exp. RAM End Address
26	XMIT Buffer Min Count
27	XMIT Buffer Max Count
28	Product ID Address (high)
30	Product ID Address (low)

Figure 3-14. OPEN Parameter List

Table 3-10. OPEN Parameter Functions

BYTE 0,1	OPEN OPTIONS. The bit functions of the Open Options field are provided below:	
	BIT 0	WRAP INTERFACE. Setting bit 0 to one on OPEN negates the Ring Insertion Process and causes all user transmit data to appear as user receive data. The data is transmitted on the lobe from the attached product to the wiring concentrator. This option can be used for system interface debug, system interface DMA testing, or lobe media testing. A **CLOSE** command must be issued to terminate WRAP mode.
	BIT 1	DISABLE HARD ERROR. If bit 1 is set to a one, the RING STATUS Hard Error and TRANSMIT BEACON Interrupts will not be generated.
	BIT 2	DISABLE SOFT ERROR. If bit 2 is set to a one, the RING STATUS Soft Error Interrupt will not be generated.
	BIT 3	PASS ADAPTER MAC FRAMES. Bit 3 is used to specify to the Adapter what action is to be taken when Adapter class MAC frames are received, but are not supported by the Adapter. If this bit is set to one the MAC frames will be passed to the attached system as normal receive data. If this bit is set to zero, the Adapter will ignore all unsupported Adapter MAC frames, purge them from internal Adapter buffers, and transmit a negative response MAC frame to the originating station.
	BIT 4	PASS ATTENTION MAC FRAMES. If bit 4 is set to one, all Attention MAC frames that are not equal to the last Attention MAC frame received will be passed to the system as normal receive data following normal processing by the Adapter.
	BIT 5	PAD ROUTING FIELD. If bit 5 is set to one, the Adapter will pad the Routing Field to 18 bytes. If no RI field is present in the received frame, the entire field will be padded to 18 bytes. This option is voided if the current buffer's data count is not at least 32 bytes. In this case the frame will be transferred as if the bit was set to zero.
	BIT 6	FRAME HOLD. If bit 6 is set to one, the Adapter will wait for an entire frame to be read from the ring before initiating the DMA transfer of the frame to the system. If this bit is a zero, then a DMA transfer will be initiated whenever an Adapter internal buffer is filled.

Table 3-10. OPEN Parameter Functions (Continued)

	BIT 7	CONTENDER. If bit 7 is set to one, the Adapter will participate in the Claim-Token Process if another Adapter detects the need for contention and initiates the Claim-Token Process. This bit has no effect if this Adapter detects the need for contention and initiates the Claim-Token Process.
	BIT 8	PASS BEACON MAC FRAMES. If this bit is set to one, the Adapter will pass Beacon MAC frames. After passing the Beacon MAC frame, the next Beacon MAC frame will be passed only if the source address or the Beacon Type Subvector changes.
	BIT 9-15	RESERVED. Bits 9-15 are ignored.
BYTES 2-7	NODE ADDRESS. This 6-byte field specifies the node address for the Adapter. If this address is all zeros, the Adapter will use the Adapter Address read from the address module. If the Node Address is not zero then the following check is made. Byte 0 bits 0,1 must be set to "01". If any of the above checks fail, the Adapter will report a Node Address Error.	
BYTES 8-11	GROUP ADDRESS. This 32-bit field specifies the Group Address and will cause the Adapter to receive messages that are sent to the Group Address. The GROUP ADDRESS can be any value. Bit 0 is ignored by the Adapter. Group Address recognition is disabled by specifying the GROUP ADDRESS as zero.	
BYTES 12-15	FUNCTIONAL ADDRESS. This 32-bit field specifies the Functional Address and will cause the Adapter to receive messages that are sent to the Functional Address. FUNCTIONAL ADDRESS bits 0, 30, and 31 are ignored by the Adapter. A zero value disables the Functional Address feature.	
BYTES 16-17	RECEIVE LIST SIZE. This 16-bit field indicates the number of bytes the Adapter will read when obtaining a Receive List from the attached system. A decimal value of 0, 14, 20, or 26 is required. If zero, the default value of 26 is used.	
BYTES 18-19	TRANSMIT LIST SIZE. This 16-bit field indicates the number of bytes the Adapter will read when obtaining a Transmit List from the attached system. A decimal value of 0, 14, 20, or 26 is required. If zero, the default value of 26 is used.	
BYTES 20-21	BUFFER SIZE. This 16-bit field indicates the Adapter's internal buffer size in bytes. BUFFER SIZE must be greater than or equal to 96. The three low-order bits must be zero. If this field is zero, a default value of 112 bytes is used.	

Table 3-10. OPEN Parameter Functions (Continued)

BYTES 22-23	EXPANSION RAM START ADDRESS. This 16-bit parameter defines an expansion RAM start address within the Adapter. This additional RAM may be used for transmit and/or receive buffers. If this field is zero, no external RAM is defined within the Adapter. If bit 15 is set to one, the existing internal Adapter RAM will not be used for transmit or receive buffers, defaulting to the expansion RAM. The expansion RAM within the Adapter begins at X'4006'and ends at X'7FFF'. The expansion RAM and decode logic are tested, but if bad parity is detected, an ADAPTER CHECK Parity Error will be issued.
BYTES 24-25	EXPANSION RAM END ADDRESS. This 16-bit field specifies the ending address of Adapter expansion RAM. The Expansion RAM End Address for the Adapter is X'7FFE'. If the EXPANSION RAM START ADDRESS is zero, this field is ignored.
BYTE 26	TRANSMIT BUFFER MINIMUM COUNT. This byte parameter contains the number of Adapter buffers that are to be reserved as transmit buffers. These buffers will be reserved for transmit only and will never be used for receive. If zero is specified, no buffers are reserved for transmit. The minimum transmit buffer count must be equal to or less than the transmit maximum buffer count (byte 27).
BYTE 27	TRANSMIT BUFFER MAXIMUM COUNT. This byte parameter contains the maximum number of Adapter buffers that are to be used for transmit. A minimum of two buffers must be available for receive. If this parameter is set to zero, a default value of six buffers is used. The product of TRANSMIT BUFFER MAXIMUM COUNT and (BUFFER SIZE -8) determines the maximum size frame that the Adapter can transmit.
BYTES 28-31	PRODUCT ID ADDRESS. This 32-bit field contains a 24-bit address of the attached system Product ID. Eighteen bytes are transferred to the Adapter during OPEN. If a Request Station Attachment MAC frame is received from the Network Manager, the bytes are read from the specified location in adapter memory. See the IBM Token-Ring Network Architecture Reference manual for more detail concerning the Product ID.

OPEN Completion Status

Upon completion of the **OPEN** command, the SSB will be loaded with the status of the OPEN completion as shown in Figure 3-15.

SSB Address + 0	X'0003'	
+ 2	Status	Error

Figure 3-15. OPEN Command SSB

The first word of the SSB contains the **OPEN** command op code. The second word contains a Status Byte and an Error Byte. The bit definitions of the Status Byte are shown in Table 3-11.

Table 3-11. OPEN Status Bit Definitions

BIT 0	ADAPTER OPEN. Bit 0 is set to one if the **OPEN** command completed successfully. All other bits will be set to zero.
BIT 1	NODE ADDRESS ERROR. Bit 1 is set to one if an error was detected in the Node Address of the Open Parameters or the Adapter Address if the node address was all zeros.
BIT 2	LIST SIZE ERROR. Bit 2 is set to one if the Receive List Size and/or the Transmit List Size are not equal to 0,14,20, or 26.
BIT 3	BUFFER SIZE ERROR. Bit 3 is set to one if BUFFER SIZE is less than 96 or negative, if the three low-order bits are not zero, or if there are less than two buffers specified.
BIT 4	EXPANSION RAM ERROR. Bit 4 is set to one if EXPANSION RAM is not within the range X'4006' to X'7FFF' or if an error (other than parity) is detected in the RAM.
BIT 5	TRANSMIT BUFFER COUNT ERROR. Bit 5 is set to one if the total number of buffers minus the TRANSMIT BUFFER COUNT is less than two.
BIT 6	OPEN ERROR. Bit 6 is set to one if an error is detected during the **OPEN** command processing. The Error byte of the SSB will specify the error.
BIT 7	RESERVED. This bit is reset to zero.

Table 3-12 specifies the Error Byte of the second word of the OPEN SSB. The Error Byte is effectively divided into two 4-bit entities (nibbles). The first 4-bit field, entitled Open Phase, is set to the **OPEN** command processing phase when the error defined in the second 4-bit field occurs. The second 4-bit field, entitled the Open Error Code, is set to the appropriate error code if a ring-related error occurs during **OPEN** command processing. Table 3-13 describes the OPEN Error codes.

Table 3-12. OPEN Phases and OPEN Error Codes

BITS 8	9	10	11	Open Command Phases
0	0	0	1	Lobe Media Test
0	0	1	0	Physical Insertion
0	0	1	1	Address Verification
0	1	0	0	Roll Call Poll
0	1	0	1	Request Initialization

BITS 12	13	14	15	Open Error Codes
0	0	0	1	Function Failure
0	0	1	0	Signal Loss
0	0	1	1	not used
0	1	0	0	not used
0	1	0	1	Timeout
0	1	1	0	Ring Failure
0	1	1	1	Ring Beaconing
1	0	0	0	Duplicate Node Address
1	0	0	1	Request Initialization
1	0	1	0	Remove Received

Table 3-13. OPEN Error Codes

FUNCTION FAILURE	This code is returned when the Adapter is unable to transmit to itself while wrapped through its lobe at the wiring concentrator, or if any MAC frames are received.
SIGNAL LOSS	This code is returned if a signal loss condition is detected at the receiver input of the Adapter during the open process (either when wrapped or inserted onto the ring).
TIMEOUT	This code is returned if the Adapter fails to logically insert onto the ring before the expiration of the insertion timer. Each phase of the insertion process must complete before the insertion time (18 seconds) expires.
RING FAILURE	This code is returned if, after becoming the Active Monitor on the ring, the Adapter times out when attempting to purge the ring. That is, the Adapter is unable to receive its own Ring Purge MAC frames.
RING BEACONING	This code is returned if the Adapter receives, after physically inserting, a Beacon MAC frame. This indicates a break in the ring.
DUPLICATE NODE ADDRESS	This code is returned if, during the Address Verification Phase, the Adapter determines that another station on the ring duplicates the Adapter's node address.
REQUEST INITIALIZATION	This code is returned if the Adapter determines that a Ring Parameter Server (RPS) is present on the ring and the RPS does not respond to a Request Initialization MAC frame. (If no RPS is present, the Adapter will not return this code.) For additional information about a Ring Parameter Server, consult the *IBM Token-Ring Network Architecture Reference Manual.*
REMOVE RECEIVED	This code is returned if the Adapter receives a Remove Adapter MAC frame during the insertion process.

Adapter Buffer Management

Frame data is transferred into the Adapter local memory before transmission on the ring. Data is stored in the Adapter local memory as a linked list of buffers. Because the chosen buffer size can affect overall Adapter performance, the local buffer size is a user-programmable option through the OPEN parameters. The default Adapter internal buffer size is 112 bytes. An example of the Adapter's internal buffer format in which the buffer size is 112 is shown in Figure 3-16. Note that the buffer size chosen for the Adapter internal buffer is independent of the data buffers used in the attached system memory. The Adapter automatically divides or combines internal memory buffers to form the minimum number of internal buffers required to represent a frame.

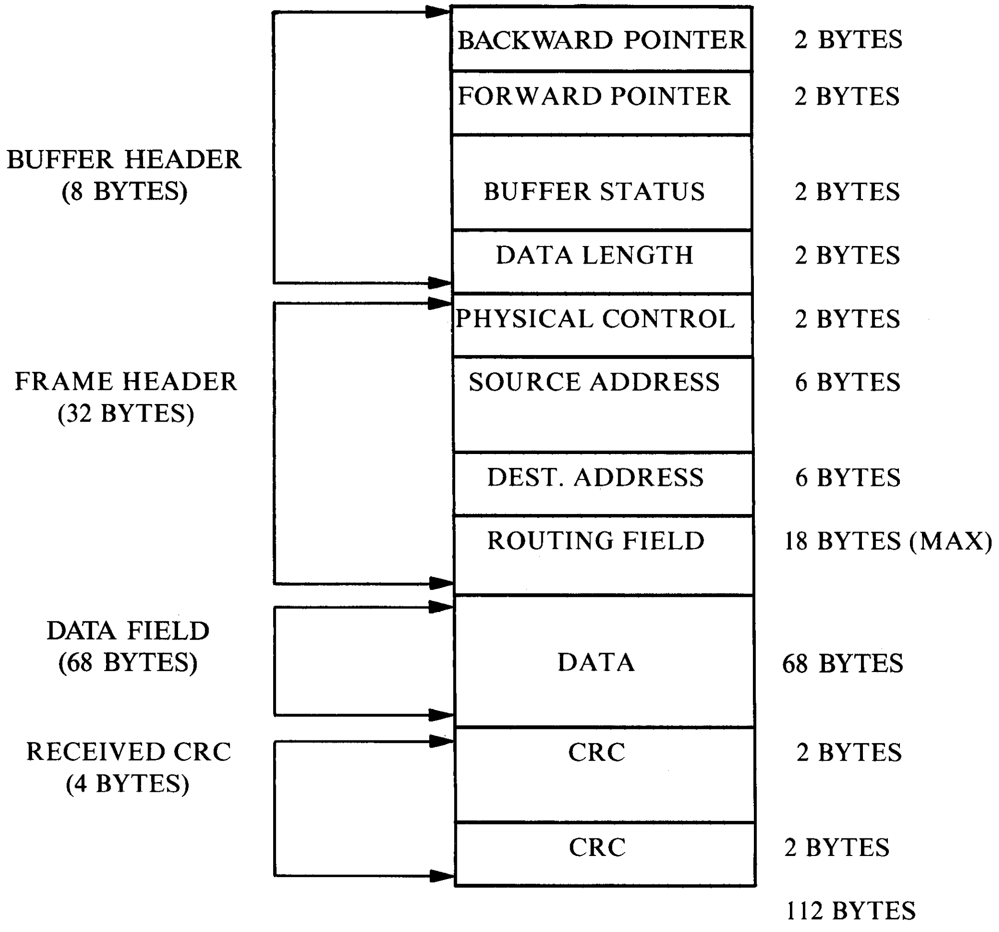


Figure 3-16. Example Adapter Internal Buffer Format

Buffer Allocation

The Adapter has 18,176 bytes of RAM available for a buffer pool for both the reception of frames from the ring media and transmission of frames to the ring media. When the attached system requests a frame transmission, buffers are taken from the buffer pool one at a time until the frame has been transferred to the Adapter. The user can specify a maximum number of these buffers to be used for transmission. The rest are dedicated for receive.

The TRANSMIT BUFFER MAXIMUM COUNT of the OPEN parameters must leave at least two buffers available to receive frames. A maximum of two transmit frames can be processed simultaneously by the Adapter. One will be enqueued for transmission while the other is transferred across the system interface, or both frames can be enqueued for transmission.

The maximum number of buffers that can be taken for frame transmission is specified in the TRANSMIT BUFFER MAXIMUM COUNT of the OPEN parameters. If the system requests transmission of a single frame that causes the number of buffers required to transmit the frame, to exceed the TRANSMIT BUFFER MAXIMUM COUNT, the **TRANSMIT** command will be terminated with a LIST ERROR of type TRANSMIT THRESHOLD. The TRANSMIT BUFFER MAXIMUM COUNT must be set such that a minimum of two receive buffers are available for receiving frames.

Buffer Size

The 18,176 bytes of RAM for a buffer pool will configure into one hundred and sixty-two 112-byte buffers. The default number of buffers is six, allowing a transmit frame maximum information field size of 600 bytes including a 32 byte frame header and 4 byte CRC field. The attached system can alter the buffer size with the **OPEN** Command BUFFER SIZE parameter.

The limit on transmit frame size is specified by the product of TRANSMIT BUFFER MAXIMUM COUNT field and (BUFFER SIZE -8) field. The buffer size must be evenly divisible by eight and a minimum of two buffers must be allocated. The minimum buffer size is 96 bytes.

TRANSMIT Command

The **TRANSMIT** command is used to transmit frames to other nodes. These frames are passed from the attached system to the Adapter using the logical format shown in Figure 3-17.

AC	FC	2 Bytes
Destination Address		6 Bytes
Source Address		6 Bytes
Routing Field		18 Bytes (max)
Data		

Figure 3-17. Attached System Frame Logical Format

Table 3-14 describes the logical frame fields.

Table 3-14. Attached System Frame Fields

AC		ACCESS CONTROL FIELD. This control field consists of the following bit functions:
	BITS 0-2	ACCESS PRIORITY. Bits 0-2 select the Access Priority for the frame. This value (0-3) must be less than or equal to the Authorized Access Priority. For a discussion of Priority, consult the *IBM Token-Ring Network Architecture Reference.*
	BITS 3-7	RESERVED. These bits are reset to zero by the Adapter.
FC		FRAME CONTROL FIELD. The bit assignments and bit definitions of the Frame Control (FC) field byte are defined in the following:
	BITS 0,1	FRAME TYPE BITS. These two bits indicate the frame type. Currently, the following frame types are defined: 00 MAC Control Frame 01 Non-MAC Control Frame 10 Reserved 11 Reserved
	BITS 2,3	RESERVED. Bits 2 and 3 are always set to zero.
	BITS 4-7	FC ATTENTION CODE. These bits indicate those frames for which the Adapter receives an internal interrupt when the frame is copied or repeated. The FC Attention code is examined in MAC frames only. Consult the *IBM Token-Ring Network Architecture Reference* for further information.

Table 3-14. Attached System Frame Fields (Continued)

DESTINATION ADDRESS	This field is 48 bits wide and contains the address of the destination. The address must be written into system memory such that the highest order byte is transferred first and the lowest order byte last. This results in the address contained within this six-byte field organized in system memory with the highest order byte occupying the lowest system byte address and the lowest order byte occupying the highest system byte address.
SOURCE ADDRESS	The Adapter will store the Node Address into the six bytes of the SOURCE ADDRESS with the exception of byte 0 bit 0 (the Routing Information Indicator). The Node Address is that address supplied by the Adapter Address or passed during the **OPEN** command.
ROUTING FIELD	The Routing Field must be included if bit 0 of the SOURCE ADDRESS field is set to one.
DATA	The Data portion is transmitted as specified by the attached system. The CRC, Ending Delimiter, and FS are appended to the data by the Adapter. Note that the attached system cannot append its own CRC.

Transmit Command Block

The **TRANSMIT** command will be rejected with Adapter COMMAND REJECT STATUS if the Adapter has not been opened, if there is already an executing **TRANSMIT** command, or if the address passed in the SCB is not aligned on an even byte boundary.

The System Command Block for a **TRANSMIT** command is shown in Figure 3-18.

SCB ADDRESS	+ 0	X'0004'
	+ 2	ADDRESS (high)
	+ 4	ADDRESS (low)

Figure 3-18. TRANSMIT SCB

ADDRESS is a 32-bit field containing a 24-bit address pointer to the Transmit Parameter List Chain. The high-order byte of the ADDRESS field is ignored. This address must be even byte-aligned. The Transmit Parameter List Chain is a 26-byte block which is used during the transmission of a single frame. A chained Transmit Parameter List is created by the attached system. It passes the first address in TRANSMIT SCB. One Transmit Parameter List cannot be used to transmit more than one frame. Several Transmit Parameter Lists can be used to transmit a single frame.

Transmit Parameter List

The Transmit Parameter List is shown in Figure 3-19.

	FORWARD POINTER
	FORWARD POINTER
	TRANSMIT CSTAT
	FRAME SIZE
1*	DATA COUNT
	DATA ADDRESS (high)
	DATA ADDRESS (low)
1*	DATA COUNT
	DATA ADDRESS (high)
	DATA ADDRESS (low)
0*	DATA COUNT
	DATA ADDRESS (high)
	DATA ADDRESS (low)

*BIT ZERO VALUE.

Figure 3-19. TRANSMIT Parameter List shown with three Data Counts

Table 3-15 describes each of the fields in the TRANSMIT Parameter List.

Table 3-15. TRANSMIT Parameter List Fields

FORWARD POINTER	This 32-bit field contains a 24-bit address which is a pointer to the next Transmit Parameter List in the chain. When this address is ODD, it denotes that the current Transmit List is the last in the chain. The Adapter will continue to process Transmit Lists until it reads an ODD address. It will then wait for the last frame (list with ODD address) to be transmitted onto the ring. If the system updates the FORWARD POINTER before the last frame is transmitted, the Adapter will continue to process the Transmit Lists. If not, the **TRANSMIT** command will complete and another must be issued to continue. The system must update the FORWARD POINTER from the most significant byte to the least significant byte to ensure that the address is valid before changing to an EVEN address. Frames, not lists that define partial frames, should be added to the chain. The FORWARD POINTER should not be initialized to point to itself, as problems may occur due to the pipelined nature of list processing employed by the Adapter. Transmit Lists must be aligned on even byte boundaries. The Adapter will not alter this parameter.
FRAME SIZE	This 16-bit field contains the number of bytes to be transmitted as a frame. The FRAME SIZE value includes AC/FC, DESTINATION and SOURCE ADDRESS, the Routing Field, and the Information Field. FRAME SIZE does not include CRC, FS, or ED. This parameter is valid only for the Transmit List that has the FRAME START bit set. FRAME SIZE must be in all lists. The Adapter will not alter this parameter. A frame size of zero is not valid. The maximum frame size which can be transmitted may be calculated as (BUFFER SIZE -8) x TRANSMIT BUFFER MAXIMUM COUNT.

Table 3-15. TRANSMIT Parameter List Fields (Continued)

DATA COUNT	This 16-bit field contains the number of bytes to be transmitted starting from the address defined in the DATA ADDRESS parameter. There can be a maximum of three DATA COUNT/DATA ADDRESS parameters to provide a gather write capability per Transmit List (not frame). If Bit 0 is zero, it is the last DATA COUNT in the Transmit List. Bit 0 of the third DATA COUNT is ignored. A DATA COUNT of 0 is permitted (with or without Bit 0 set). The sum of the used DATA COUNT parameters must equal the FRAME SIZE specified on the Start of Frame List. The DATA COUNT can be even or odd. The Adapter will not alter this parameter.
DATA ADDRESS	This 32-bit field contains the 24-bit address of the data to be transmitted. DATA ADDRESS may be even or odd. The Adapter will not alter this parameter.
TRANSMIT CSTAT	TRANSMIT COMMAND/STATUS. This 16-bit parameter is set by the attached system when the Transmit List is created. It is over-written by the Adapter to report frame completion status. When initially set by the attached system, this parameter field is referred to as the TRANSMIT CSTAT REQUEST field. After a frame completes transmission, the Adapter will overwrite bits in this field only in the list which starts the frame. These bits indicate the completion status of the frame. This parameter field is referred to as the TRANSMIT CSTAT COMPLETE. Note that command and status information within the TRANSMIT CSTAT field is associated with frames and not the **TRANSMIT** command directly. TRANSMIT CSTAT REQUEST. The CSTAT bits are set by the attached system as follows: BIT 0 — VALID. The Adapter will wait for bit 0 to be set to a one before processing the current Transmit List. The attached system must issue a TRANSMIT VALID Interrupt Request when changing VALID bits from a zero to a one. This bit is ignored unless the list is an anticipated start of frame (i.e. follows End of Frame or is first list of command).

Table 3-15. TRANSMIT Parameter List Fields (Continued)

	BIT 1	FRAME COMPLETE. Bit 1 should be reset to zero.
	BIT 2	START OF FRAME. Bit 2 must be set to one for a list which defines the start of a frame.
	BIT 3	END OF FRAME. Bit 3 must be set to one for a list which defines the end of a frame.
	BIT 4	FRAME INTERRUPT. Setting bit 4 to one will cause the Adapter to interrupt when the frame has been transmitted, rather than waiting for all frames on the chain to be transmitted. This bit is ignored unless START OF FRAME (bit 2) is a one.
	BIT 5-15	RESERVED. Bits 5-15 should be set to zero.
TRANSMIT CSTAT COMPLETE.	This 16-bit field is loaded, on a list which starts a frame only, with the completion code for the transmitted frame (not **TRANSMIT** command) when the Adapter has transmitted a frame. CSTATs which are not in a list which defines the start of a frame, are not altered by the Adapter. The TRANSMIT CSTAT COMPLETE bit definitions are shown below:	
	BIT 0	VALID. Bit 0 is reset to zero.
	BIT 1	COMPLETE. Bit 1 is set to one.
	BIT 2	FRAME START. Bit 2 is set to one.
	BIT 3	FRAME END. Bit 3 is the same as specified in CSTAT REQUEST.
	BIT 4	INTERRUPT (FRAME). Bit 4 is the same as specified in CSTAT REQUEST.
	BIT 5	TRANSMIT ERROR. Bit 5 is set to one if the frame transmit or strip process was in error.

Table 3-15. TRANSMIT Parameter List Fields (Continued)

	BITS 6-7	RESERVED. Bits 6-7 are the same as that specified in the CSTAT REQUEST.
	BIT 8-15	STRIP FS. Bits 8-15 contain a copy of the FRAME STATUS Field (FS) byte returned when the transmitted frame is stripped off the ring. If TRANSMIT ERROR is set, the FS should be ignored. Bits 14 and 15 of the stripped FS will be set to zero. For further discussion concerning FS consult the *IBM Token-Ring Network Architecture Reference.*

Transmit Completion

An interrupt will be generated for the **TRANSMIT** command when:

- All the frames specified by the Transmit Parameter List Chain have been transmitted, or;
- The **TRANSMIT HALT** command has completed, or;
- A frame has been transmitted that had FRAME INTERRUPT set in CSTAT, or;
- A list error is detected.

The SSB will be loaded as shown in Figure 3-20.

SSB ADDRESS	+ 0	X‘0004’
	+ 2	XMIT COMPLETE
	+ 4	LIST ADDRESS (high)
	+ 6	LIST ADDRESS (low)

Figure 3-20. TRANSMIT SSB

The Transmit Complete Field bit definitions are provided in Table 3-16.

Table 3-16. Transmit Complete Field Bit Definitions

BIT 0	COMMAND COMPLETE. Bit 0 is set to one to indicate that the **TRANSMIT** command has completed. The system must issue another **TRANSMIT** command to transmit additional frames. LIST ADDRESS will contain the address of the last Transmit List processed. This bit is also set as a result of a **TRANSMIT HALT** command. If a **TRANSMIT HALT** is issued and no frames have been transmitted, LIST ADDRESS will be cleared. The COMMAND COMPLETE and FRAME COMPLETE bits are not set at the same time.
BIT 1	FRAME COMPLETE. Bit 1 is set to one to indicate that a frame has been transmitted and the FRAME INTERRUPT bit was set in CSTAT. Since frames on the Transmit Chain can be transmitted faster than the system can respond to the interrupts and/or faster than the Adapter can cause the interrupts, the FRAME COMPLETE interrupt can report the completion of more than one frame at a time. LIST ADDRESS will contain the address of the last Transmit List of the last frame transmitted. If lists with the FRAME INTERRUPT set are intermixed with lists that do not have FRAME INTERRUPT set, FRAME COMPLETE can include frames that did not have FRAME INTERRUPT set.
BIT 2	LIST ERROR. Bit 2 is be set to one if there is an error in one of the lists that comprise a frame. Bits 8-13 define the error. The **TRANSMIT** command will be terminated and the system must issue another **TRANSMIT** command to continue transmission. LIST ADDRESS will contain the address of the list that starts the frame with parameter errors. The LIST ERROR interrupt will not occur until all other transmit status has been posted. The CSTATs of lists found to be in error are not altered by the Adapter. Neither the FRAME COMPLETE nor the COMMAND COMPLETE bits will be set with LIST ERROR.
BIT 3-7	RESERVED. Bits 3-7 are reset to zero.
BIT 8	FRAME SIZE. Bit 8 is set to one if FRAME SIZE does not equal the sum of the DATA COUNTs or if the frame size is less than the required header plus one byte of Information Field (15 bytes plus Routing Field), or a frame size of zero was specified.

Table 16. Transmit Complete Field Bit Definitions (Continued)

BIT 9	TRANSMIT THRESHOLD. Bit 9 is set to one if a single frame size determined by (BUFFER SIZE -8) x TRANSMIT BUFFER MAXIMUM COUNT, as specified in the **OPEN** command.
BIT 10	ODD ADDRESS. Bit 10 is set to one if an odd FORWARD POINTER is read on a list that is not Frame End.
BIT 11	FRAME START. Bit 11 is set to one if the FRAME START bit is set to one on a list that is not an anticipated start of frame or the FRAME START bit is not set to one on an anticipated start of frame.
BIT 12	UNAUTHORIZED ACCESS PRIORITY. Bit 12 is set to one if the Access Priority requested has not been authorized.
BIT 13	UNAUTHORIZED MAC FRAME. Bit 13 is set to one if the Adapter is not authorized to send a MAC Frame with the Source Class specified, if the MAC frame has a source class of zero or if the MAC frame FC ATTN field is greater than one. For more information concerning MAC frames, consult the *IBM Token-Ring Network Architecture Manual.*
BIT 14	ILLEGAL FRAME FORMAT. If bit 0 of the FC field was set to one, the **TRANSMIT** command will terminate with bit 14 set to one.
BIT 15	RESERVED. Bit 15 is reset to zero.

The attached system can create a chain of a fixed number of Transmit Lists, set the last list FORWARD POINTER to the address of the first list and manipulate the VALID bits to initiate transmission. When the Adapter reads a Frame Start List with the VALID bit reset to zero, it will suspend processing until a TRANSMIT VALID Interrupt Request is issued by the system. The system is not notified of this condition. The TRANSMIT VALID Interrupt must be issued when changing one or more VALID bits from zero to one when the list is on the Transmit Chain.

The TRANSMIT VALID interrupt can be issued at any time and the Adapter will ignore the interrupt if it is not waiting for a VALID bit transition.

If a fixed Transmit Chain technique is utilized and more than one list is used to transmit a single frame, lists that do not have the FRAME START bit set should have the VALID bit reset to zero. Since the Adapter does not alter the CSTAT field for lists that are not Start of Frame, re-validating of the start of frame list will also release the remaining frame lists if the VALID bits were initially set.

A FORWARD POINTER should not be set to point to itself. That is, a chain should not be made with one transmit list.

Due to the pipelined nature of transmit list processing, (i.e. the first frame is transmitted while the second frame is concurrently DMAed from system memory) this technique can cause the Adapter to erroneously send the same frame twice.

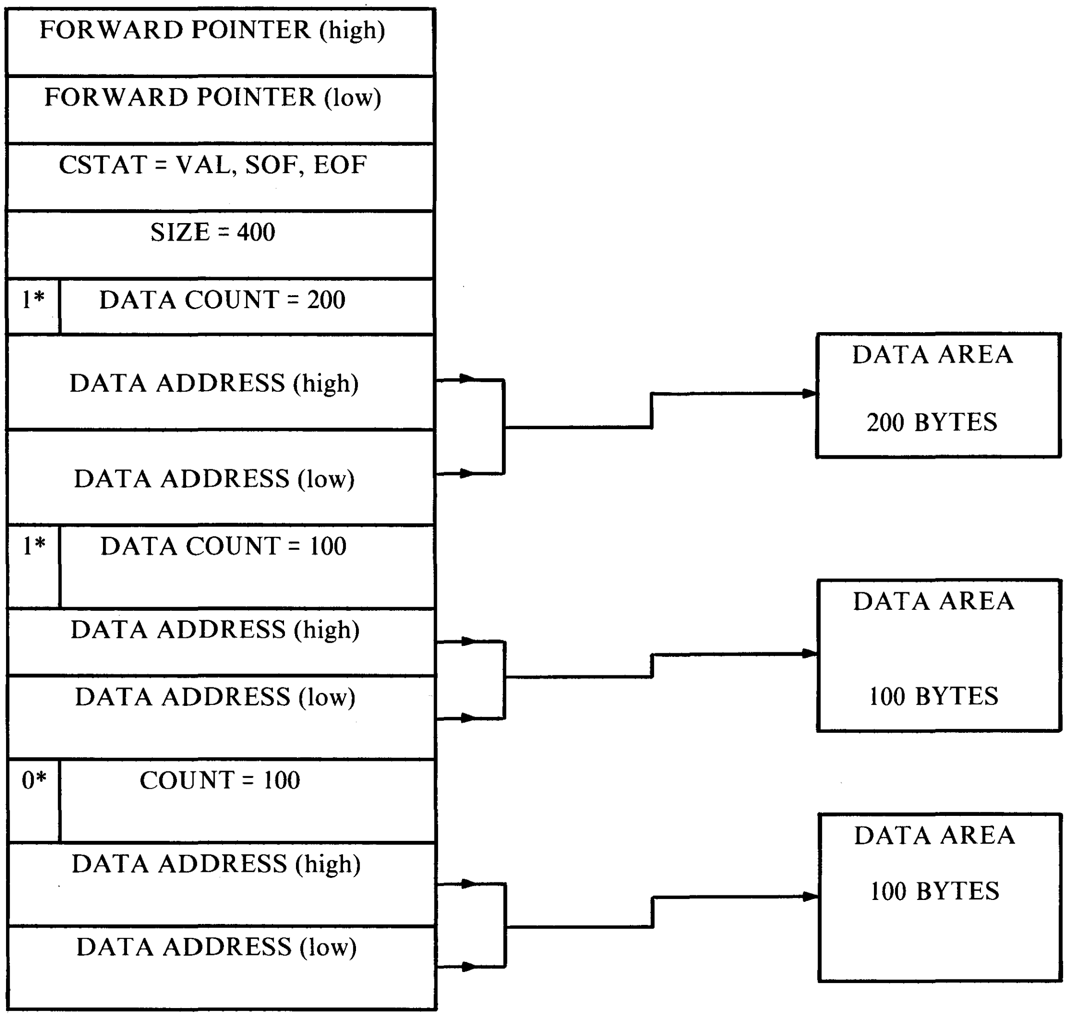
The DATA ADDRESS parameters in the Transmit List can be on even or odd byte boundaries. If the Adapter is to read data from an even byte system address to an internal odd byte address (due to an odd Data Count), it will transfer a single byte and transfer the remaining data starting at an odd system address.

Since Transmit Lists can be added dynamically to the Transmit Chain, a test should be made following COMMAND COMPLETE to determine if the Adapter has processed all frames that the attached system has placed on the chain. If frames have been added to the chain subsequent to the **TRANSMIT** command, the FORWARD POINTER at the address contained in LIST ADDRESS should be examined following COMMAND COMPLETE.

If the FORWARD POINTER is ODD, all frames have been transmitted. If the FORWARD POINTER is EVEN, another **TRANSMIT** command should be executed with the SCB pointer to the Transmit List Chain set equal to that FORWARD POINTER.

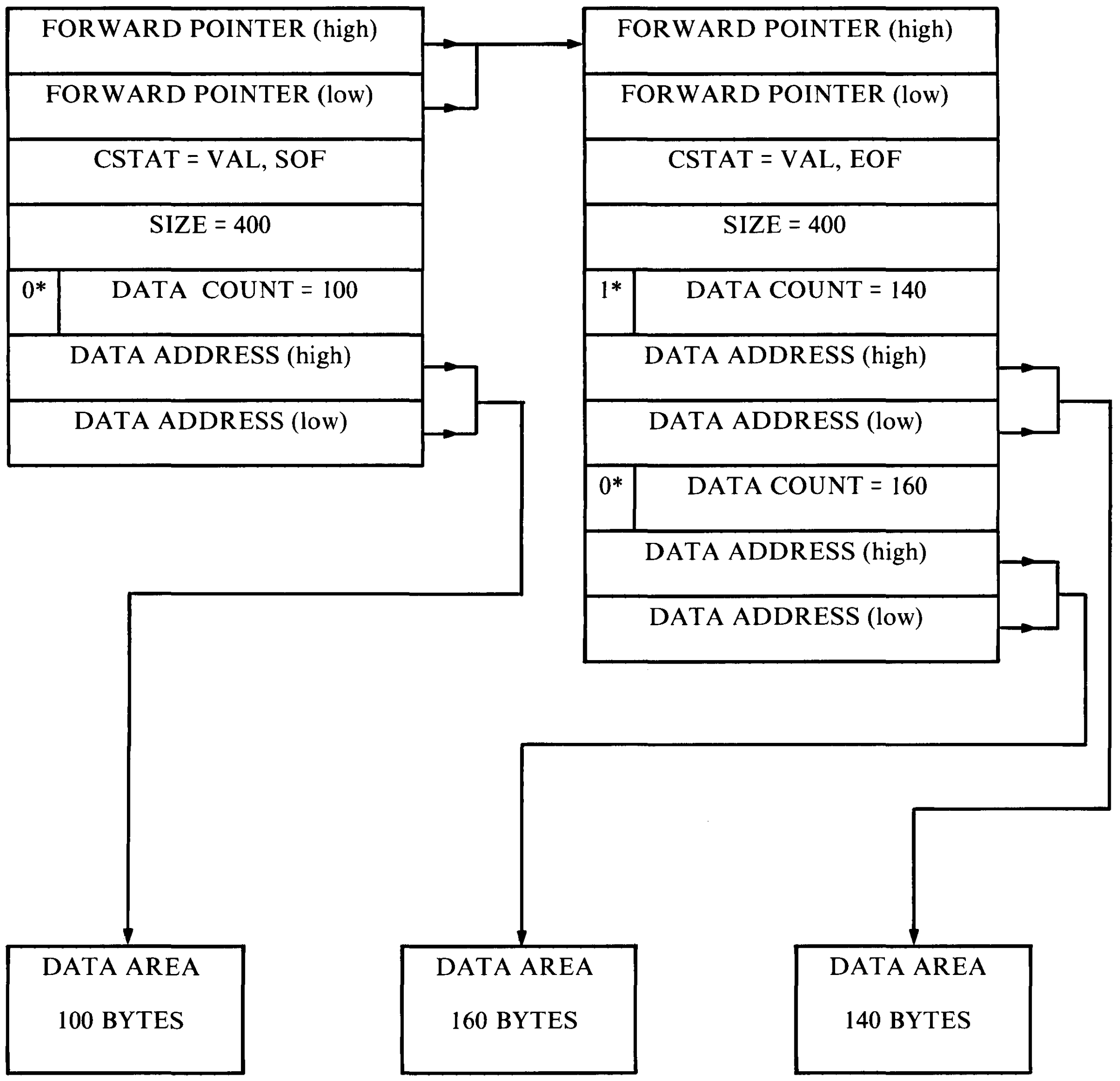
Transmit List Examples

Four examples (Figures 3-21 to 3-24) of possible list formats will be illustrated and all result in the transmission of a single 400-byte frame. Figure 3-23 and Figure 3-24 are configured such that the attached system buffer space is appended to a 14-byte list.



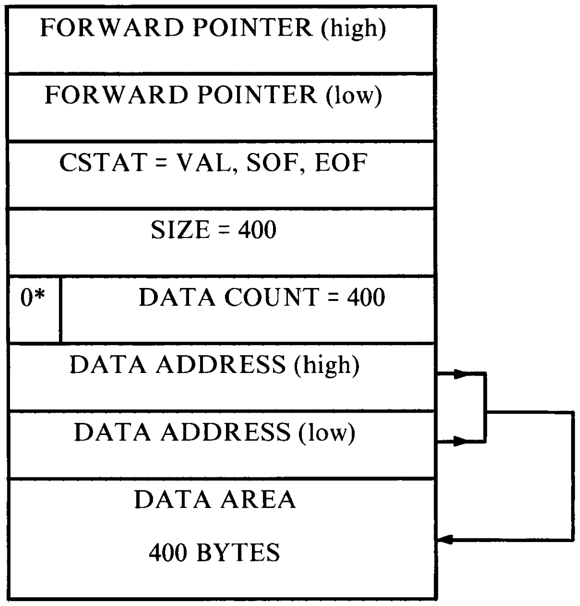
*Bit Zero Value.

Figure 3-21. Transmit List Format: Example 1



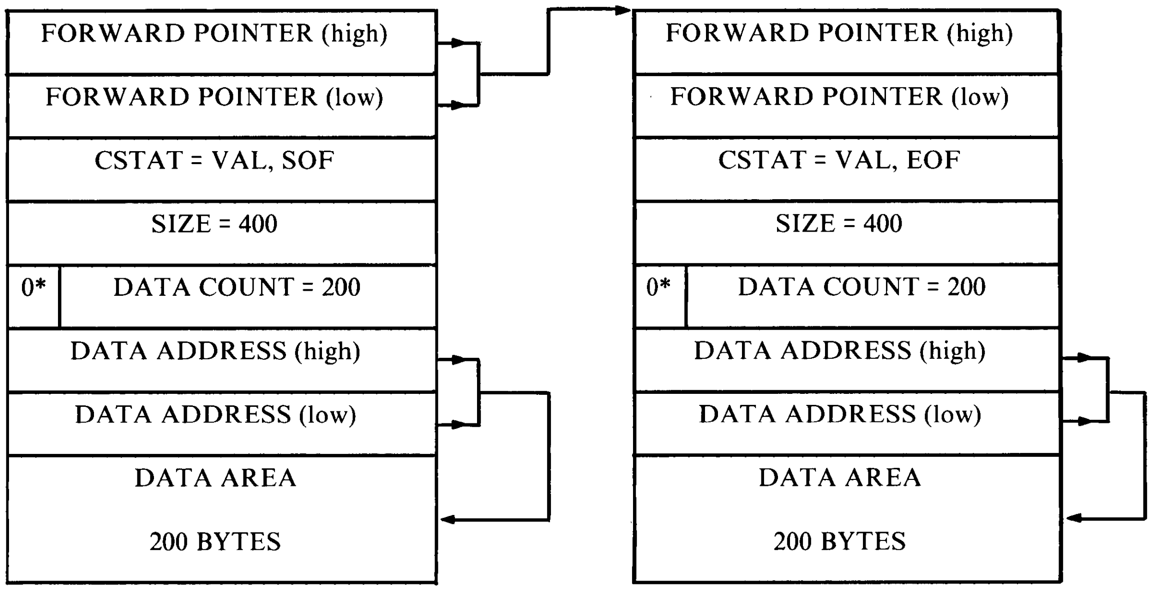
*Bit Zero Value

Figure 3-22. Transmit List Format: Example 2



*Bit Zero Value.

Figure 3-23. Transmit List Format: Example 3



*Bit Zero Valve.

Figure 3-24. Transmit List Format: Example 4

TRANSMIT HALT Command

The **TRANSMIT HALT** command is used to interrupt the Transmit List chain. Following recognition of this command, the Adapter will terminate the transmit chain as soon as possible. Any frames queued in the Adapter will be purged and the **TRANSMIT** command will be terminated with COMMAND COMPLETE status. If there is not an executing **TRANSMIT** command, **TRANSMIT HALT** is ignored.

TRANSMIT HALT Command Block

The SCB for a **TRANSMIT HALT** command is shown in Figure 3-25.

SCB ADDRESS	+ 0	X'0005'
	+ 2	———
	+ 4	———

Figure 3-25. TRANSMIT HALT Command SCB

The 32-bit field following the COMMAND field of the SCB is read by the Adapter, but ignored.

RECEIVE Command

The **RECEIVE** command is used to receive frames from other stations on the ring. This command normally is issued only once (after OPEN), since receive data is dynamically added to a Receive Parameter List Chain. The **RECEIVE** command can be terminated due to a list error.

The logical format of received frames passed across the system interface is identical to the logical format of transmit frames, as shown in Figure 3-17.

The AC and FC, destination address and source address fields, are transferred to the attached system as they were received from the ring.

The Routing Field is passed to the attached system for all frames. If the PAD ROUTING FIELD option is specified during OPEN, the routing field will be padded to 18 bytes. If the frame does not contain a Routing Field the field will still be padded to 18 bytes. The padding will not alter the contents of the systems data buffer.

The **RECEIVE** command will be rejected with Adapter COMMAND REJECT STATUS under the following conditions:

- If the Adapter has not been opened, or;
- If there is already an executing **RECEIVE** command, or;
- If the address passed in the SCB is not word aligned.

RECEIVE Command Block

The SCB for a **RECEIVE** command is shown in Figure 3-26.

SCB ADDRESS	+ 0	X‘0006’
	+ 2	ADDRESS (high)
	+ 4	ADDRESS (low)

Figure 3-26. RECEIVE Command SCB

The 32-bit ADDRESS field contains a 24-bit address pointer to a Receive Parameter List Chain. The high-order byte of this field is ignored. This address must be aligned on even byte boundary.

RECEIVE Parameter List

The attaching system creates a chained Receive Parameter List as shown in Figure 3-27. The address of the Receive Parameter list is in the **RECEIVE** command. A single Receive Parameter List cannot be used to receive more than one frame. Several Receive Parameter Lists can be used to receive a single frame.

The Receive Parameter List is a 14, 20 or 26-byte data structure as shown in Figure 3-27.

	FORWARD POINTER (high)
	FORWARD POINTER (low)
	RECEIVE CSTAT
	FRAME SIZE
1*	DATA COUNT
	DATA ADDRESS (high)
	DATA ADDRESS (low)
1*	DATA COUNT
	DATA ADDRESS (high)
	DATA ADDRESS (low)
0*	DATA COUNT
	DATA ADDRESS (high)
	DATA ADDRESS (low)

*Bit Zero Value.

Figure 3-27. RECEIVE Parameter List shown with three Data Counts

Table 3-17 describes each of the fields in the RECEIVE Parameter List.

Table 3-17. RECEIVE Parameter List Field Definitions

PARAMETER	DEFINITION
FORWARD POINTER	This 32-bit field contains a 24-bit address pointer to the next RECEIVE Parameter List in the chain. When this address is ODD, it denotes that the current RECEIVE Parameter List is the last in the chain. The Adapter will DMA write a received frame into the address(es) specified in the RECEIVE List and then check the FORWARD POINTER. If it is odd, the Adapter will interrupt the system with a request to place additional lists on the chain. The Adapter will not terminate the **RECEIVE** command and will wait for a RECEIVE CONTINUE Interrupt Request to resume the receive operation. The attached system must update the FORWARD POINTER from the most significant byte to the least significant byte to ensure that the address is valid before changing to an EVEN address. Receive Lists must be aligned on 16-bit boundaries. The Adapter will not alter this parameter.
FRAME SIZE	This 16-bit field contains the number of bytes in the received frame. The Adapter will store this count in the Receive List which starts a new frame. FRAME SIZE includes AC, FC, Destination and Source Address, Routing Field (if any), pad length (if PAD ROUTING FIELD specified) and the Data Field. FRAME SIZE does not include CRC, FS, or ED.
DATA COUNT	This 16-bit field contains the maximum number of bytes that can be stored starting at the address defined in the DATA ADDRESS parameter. There can be a maximum of three DATA COUNT/ DATA ADDRESS parameters to provide a scatter read capability per Received List (not frame). If bit 0 is cleared, it is the last DATA COUNT in the Receive List. Bit 0 of the third DATA COUNT is ignored. A zero DATA COUNT is permitted (with or without Bit 0 set). The DATA COUNT can be even or odd. The Adapter will not alter this parameter.

Table 3-17. RECEIVE Parameter List Field Definitions (Continued)

	If the PAD ROUTING FIELD option is specified during the **OPEN** command, then the first DATA COUNT in a Receive List used for start of frame must be at least 32. This allows space for the AC, FC, DESTINATION ADDRESS, SOURCE ADDRESS, and the ROUTING FIELD which will be padded to 18 bytes. If the DATA COUNT is less than 32, the option will be voided.
DATA ADDRESS	This 32-bit field contains the 24-bit address of the data to be received. DATA ADDRESS may be even or odd. The Adapter will not alter this parameter.
RECEIVE CSTAT	RECEIVE COMMAND/STATUS. This 16-bit parameter is set by the attached system when the Receive List is created and is over-written by the Adapter to report frame completion status. When initially set by the attached system, this parameter field is referred to as the RECEIVE CSTAT REQUEST field. After a receipt of a frame, the Adapter will overwrite bits in this field only in the lists which start or end a frame. These bits indicate the completion status of the frame. This parameter field is referred to as the RECEIVE CSTAT COMPLETE. Note that command and status information within the RECEIVE CSTAT field is associated with frames and not directly with the **RECEIVE** command.
	RECEIVE CSTAT REQUEST. The RECEIVE CSTAT REQUEST bit definitions are set by the attached system as shown below:

BIT 0	VALID. The Adapter will wait for bit 0 to be set to one before placing data in the current Receive List. A RECEIVE VALID Interrupt Request must be issued by the attaching system when changing VALID bits. This bit is examined for every Receive List.
BIT 1	COMPLETE. Bit 1 should be reset to zero.
BIT 2	FRAME START. Bit 2 should be reset to zero.
BIT 3	FRAME END. Bit 3 should be reset to zero.

Table 3-17. RECEIVE Parameter List Field Definitions (Continued)

BIT 4	INTERRUPT (FRAME). Bit 4, when set to one, will cause the Adapter to interrupt when the frame has been received. This bit is ignored unless the FRAME START bit is set.
BIT 5	INTERFRAME WAIT. Setting this bit to one will cause the Adapter to interrupt when a frame has been received. The Adapter will assume a receive suspended state waiting for the RECEIVE CONTINUE bit of the interrupt Register to be set to one prior to resuming frame transfer. The next list to be used is addressed by the FORWARD POINTER of the list which has the END OF FRAME bit set. This bit is ignored in lists which are not start of frame lists (bit 2 is not set to one).
BITS 6-15	RESERVED. Bits 6-15 should be reset to zero.

RECEIVE CSTAT COMPLETE When a frame has been transferred to the system, the CSTATs for the Lists which start and end a frame are updated by the Adapter as follows:

BIT 0	VALID. Bit 0 is reset to zero.
BIT 1	FRAME COMPLETE. Bit 1 is set to one.
BIT 2	FRAME START. Bit 2 is set to one on the list that starts a frame.
BIT 3	FRAME END. Bit 3 is set to one on the list that ends a frame.
BITS 4-7	RESERVED. Bits 4-7 are reset to zero.
BITS 8-13	RECEIVED FS. When a FRAME START bit is reset to zero, these bits will also be reset to zero. When FRAME START is set to one, these bits will contain the high order 6-bits of the received FS.
BITS 14-15	RESERVED.

RECEIVE Completion

An interrupt will be generated for the **RECEIVE** command when the Receive Parameter List chain has ended (odd address in FORWARD POINTER) or when a frame is copied into a list that has Frame Interrupt set in the CSTAT parameter.The SSB upon receive completion will be loaded as shown in Figure 4-28.

SSB ADDRESS	+ 0	X'0006'
	+ 2	RECV COMPLETE
	+ 4	LIST ADDRESS (high)
	+ 6	LIST ADDRESS (low)

Figure 3-28. RECEIVE Command SSB

The RECEIVE COMPLETE Field bit definitions are provided in Table 3-18.

Table 3-18. RECEIVE COMPLETE Field Bit Definitions

BIT 0	FRAME COMPLETE. Bit 0 is set to one to indicate that a frame has been received and the FRAME INTERRUPT or INTERFRAME WAIT bit was set in CSTAT. Since frames can be received and transferred faster than the attached system can respond to the interrupts and/or faster than the Adapter can cause the interrupts, the FRAME COMPLETE interrupt can report the completion of more than one frame at a time. The 32-bit LIST ADDRESS will contain the address of the last Receive List of the last frame transferred to the system. If lists with the FRAME INTERRUPT bit set are intermixed with lists that do not have FRAME INTERRUPT set, FRAME COMPLETE can include frames that did not have FRAME INTERRUPT set. FRAME COMPLETE will not be set with the RECEIVE SUSPENDED bit also set.
BIT 1	RECEIVE SUSPENDED. Bit 1 is set to one when the Adapter detects an odd address in the Receive Parameter List Chain. LIST ADDRESS will contain the address of the list that has an odd FORWARD POINTER. The attached system must update the FORWARD POINTER and issue a RECEIVE CONTINUE Interrupt Request in order to continue. RECEIVE SUSPENDED will not be set with the FRAME COMPLETE bit also set.
BITS 2-15	RESERVED. Bits 2-15 are reset to zero.

The examples (Figure 3-21 to Figure 3-24) for Transmit Parameter List Formats can also be applied to Receive Parameter List Formats.

An attached system can create a chain of a fixed number of Receive Parameter Lists by setting the last list FORWARD POINTER to the address of the first list, and manipulate the VALID bits appropriately to initiate reception.

When the Adapter reads any list with VALID reset to zero, it will suspend processing until a RECEIVE VALID Interrupt Request is issued by the system. The attached system is not notified of this condition. The RECEIVE VALID Interrupt must be issued when changing the condition of one or more VALID bits from 0 to 1 when the list is on the Receive Chain.

The RECEIVE VALID Interrupt can be issued at any time and the Adapter will ignore the interrupt if it is not waiting for a transition in the VALID bit.

If a fixed Receive Chain technique is utilized and more than one list is used to receive a single frame, caution should be exercised when validating the lists. Since the Adapter does not alter the CSTAT for lists that are not the start of a frame or the end of a frame, re-validation of the start of frame list would also release the middle of frame lists.

The DATA ADDRESS parameters in the Receive List can be an even or odd byte alignment.

If the Adapter is to write data to an even system address from an internal odd address (i.e. odd DATA COUNT), then it will transfer a single byte and then transfer the remaining data starting at an odd system address.

The RECEIVE CONTINUE Interrupt can be issued at any time and the Adapter will ignore the interrupt if it is not waiting for a FORWARD POINTER transition from an odd to even address.

Header Routing

Some systems may need to receive only a frame header (or header and a portion of the data) and route the remainder of the frame data according to the contents of the header. This can be accomplished as follows:

1. Set FRAME HOLD in the **OPEN** Command.
2. Post a Receive List that has an odd FORWARD POINTER and one DATA COUNT/DATA ADDRESS parameter sufficient to hold the desired header.
3. The Adapter will use the list and interrupt the system with RECEIVE SUSPENDED, leaving the CSTAT unchanged. (Should the entire frame be less than or equal to DATA COUNT, a FRAME COMPLETE interrupt will be posted if requested.)

4. Following RECEIVE SUSPENDED, the system can examine the header and determine the frame destination. The frame size field of the receive parameter list will not be updated by the Adapter and is not valid.
5. Post additional lists to receive the data by writing a non-odd address in the FORWARD POINTER field, followed by another header list with an odd FORWARD POINTER.
6. Issue a RECEIVE CONTINUE Interrupt Request.
7. When the frame has been transferred, a FRAME COMPLETE interrupt will occur (if requested).

CLOSE Command

The **CLOSE** command is used to terminate transmission on the ring or to terminate **OPEN** with the wrap option, and will purge all frames in the Adapter. The **CLOSE** command will be rejected with COMMAND REJECT STATUS if the Adapter is not open.

CLOSE Command Block

The SCB for a **CLOSE** command is shown in Figure 3-29.

SCB ADDRESS	+ 0	X'0007'
	+ 2	———
	+ 4	———

Figure 3-29. CLOSE Command SCB

The 32-bit field following the COMMAND field is read by the Adapter, but ignored.

CLOSE Completion

Upon close completion the SSB will be loaded with Close Completion status as shown in Figure 3-30.

SSB ADDRESS	+ 0	X'0007'
	+ 2	CLOSE COMPLETION

Figure 3-30. CLOSE Command SSB

Table 3-19 describes the CLOSE COMPLETION field bit definitions.

Table 3-19. CLOSE COMPLETION Field Bit Definitions

BIT 0	ADAPTER CLOSED. Bit 0 is set to one when the **CLOSE** command is completed.
BITS 1-15	RESERVED. Bits 1-15 are reset to zero.

SET GROUP ADDRESS Command

The **SET GROUP ADDRESS** command is used to set the Adapter Group Address if it is to be changed after **OPEN**. The **SET GROUP ADDRESS** command will be rejected with COMMAND REJECT STATUS if the Adapter is not open.

SET GROUP ADDRESS Command Block

The SCB for a **SET GROUP ADDRESS** command is shown in Figure 3-31.

SCB ADDRESS	+ 0	X'0008'
	+ 2	GROUP ADDRESS (high)
	+ 4	GROUP ADDRESS (low)

Figure 3-31. SET GROUP ADDRESS Command SCB

The 32-bit address following the COMMAND field in the SCB is the Group Address and is stored in the Adapter Group Address Register. Bit zero is ignored.

SET GROUP ADDRESS Completion

Upon completion of the **SET GROUP ADDRESS** command, the SSB will be loaded with **SET GROUP ADDRESS** completion status as depicted in Figure 3-32.

SSB ADDRESS	+ 0	X'0008'
	+ 2	SET GRP ADDR COMP

Figure 3-32. SET GROUP ADDRESS Command SSB

Table 3-20 describes the SET GROUP ADDRESS completion field bit definitions.

Table 3-20. SET GROUP ADDRESS Completion Field Bit Definitions

BIT 0	COMMAND COMPLETE. Bit 0 is set when the SET GROUP ADDRESS command is completed.
BITS 1-15	RESERVED. Bits 1-15 are reset to zero.

SET FUNCTIONAL ADDRESS Command

The **SET FUNCTIONAL ADDRESS** command is used to set and reset the Adapter Functional Address if it is to be changed after **OPEN**. The **SET FUNCTIONAL ADDRESS** command will be rejected with COMMAND REJECT STATUS if the Adapter is not open.

SET FUNCTIONAL ADDRESS Command Block

The SCB for a **SET FUNCTIONAL ADDRESS** command is shown in Figure 3-33.

SCB ADDRESS	+ 0	X'0009'
	+ 2	FUNCTIONAL ADDR (high)
	+ 4	FUNCTIONAL ADDR (low)

Figure 3-33. SET FUNCTIONAL ADDRESS Command SCB

The 32-bit address following the COMMAND field in the SCB is the Functional Address and is stored in the Adapter's internal Functional Address Register. Bits 30 and 31 of the Functional Address are ignored (corresponding to the Active Monitor and Ring Parameter Server). Bit 0 (most significant bit) is also ignored.

SET FUNCTIONAL ADDRESS Completion

Upon completion of the **SET FUNCTIONAL ADDRESS** command, the SSB will be loaded with SET FUNCTIONAL ADDRESS completion status as depicted in Figure 3-34.

SSB ADDRESS	+ 0	X'0009'
	+ 2	SET FTN ADDR COMP

Figure 3-34. SET FUNCTIONAL ADDRESS Command SSB

Table 3-21 describes the SET FUNCTIONAL ADDRESS Completion field bit definitions.

Table 3-21. SET FUNCTIONAL ADDRESS Completion Field Bit Definitions

BIT 0	COMMAND COMPLETE. Bit 0 is set when the **SET FUNCTIONAL ADDRESS** command is completed.
BITS 1-15	RESERVED. Bits 1-15 are reset to zero.

READ ERROR LOG

The **READ ERROR LOG** command is used to read and reset the Adapter Attached Product Error Log. After **READ ERROR LOG** command completion, the Error Log will be all zeros.

READ ERROR LOG Command Block

The SCB for a **READ ERROR LOG** command is shown in Figure 3-35.

SCB ADDRESS	+ 0	X'000A'
	+ 2	ADDRESS (high)
	+ 4	ADDRESS (low)

Figure 3-35. READ ERROR LOG Command SCB

The 32-bit ADDRESS field contains a 24-bit starting address location where the 14-byte Error Log will be written in attached system memory. The 14-byte Error Log table is shown in Figure 3-36.

BYTE + 0	LINE ERROR	RESERVED
+2	BURST ERROR	ARI/FCI ERROR
+4	RESERVED	RESERVED
+6	LOST FRAME ERROR	RECEIVE CONGESTION ERROR
+ 8	FRAME COPIED ERROR	RESERVED
+10	TOKEN ERROR	RESERVED
+12	DMA BUS ERRORS	DMA PARITY ERRORS

Figure 3-36. Error Log Table

READ ERROR LOG Completion

Upon completion of the **READ ERROR LOG** command, the SSB will be loaded with **READ ERROR LOG** completion status as shown in Figure 3-37.

SCB ADDRESS + 0	X'000A'
+ 2	ERROR LOG COMPLETION

Figure 3-37. READ ERROR LOG Command SSB

Table 3-22 describes the READ ERROR LOG COMPLETION field bit definitions.

Table 3-22. READ ERROR LOG COMPLETION Field Bit Definitions

BIT 0	COMMAND COMPLETE. Bit 0 is set to one when the READ ERROR LOG command is completed.
BITS 1-15	RESERVED. Bits 1-15 are reset to zero.

READ ADAPTER BUFFER Command

The **READ ADAPTER BUFFER** command is used to transfer Adapter storage across the system interface to attached system memory.

READ ADAPTER BUFFER Command Block

The SCB for a **READ ADAPTER BUFFER** command is shown in Figure 3-38.

SCB ADDRESS	+ 0	X'000B'
	+ 2	ADDRESS
	+ 4	ADDRESS

Figure 3-38. READ ADAPTER BUFFER Command SCB

The 32-bit ADDRESS field contains a 24-bit address pointer to buffer space in the attached system memory. The system buffer space is shown in Figure 3-39.

+ 0	DATA COUNT
+ 2	DATA ADDRESS
+ 4	DATA AREA

Figure 3-39. READ ADAPTER System Buffer Space

The 16-bit DATA COUNT field contains the number of bytes to read from the Adapter. The data will be stored starting at the DATA COUNT location which is at the beginning of the system's buffer space.

The DATA ADDRESS is a 16-bit field containing the address of the data in the Adapter to be read. Bit 15 is reset to zero by the Adapter. The DATA ADDRESS is not checked for valid extents. The **READ ADAPTER** command will result in an ADAPTER CHECK Parity Error if reference is made to an undefined storage area.

Table 3-23 illustrates some of the internal Adapter pointers accessible via the **READ ADAPTER BUFFER** command. These pointers will reside beginning at location X'0A00' in Adapter memory. These pointers must be read following initialization but before issuing an **OPEN** command.

Table 3-23. Adapter Internal Pointers

Address	Description
X'0A00'	Pointer to Internal Adapter Address
X'0A02'	Pointer to Software Level
X'0A04'	Pointer to Adapter addresses:
	pointer + 0 Node Address
	pointer + 6 Group Address
	pointer + 10 Functional Address
X'0A06'	Pointer to Adapter Parameters
	pointer + 0 Physical Drop Number
	pointer + 4 Upstream Node Address
	pointer + 10 Upstream Physical Drop Number
	pointer + 14 Last Poll Address
	pointer + 22 Transmit Access Priority
	pointer + 24 Source Class Authorization
	pointer + 26 Last Attention Code
	pointer + 28 Last Source Address
	pointer + 34 Last Beacon Type
	pointer + 36 Last Major Vector
	pointer + 38 Ring Status
	pointer + 40 Soft Error Timer Value
	pointer + 42 Ring Interface Error Counter
	pointer + 44 Reserved
	pointer + 46 Monitor Error Code
	pointer + 48 Beacon Transmit Type
	pointer + 50 Beacon Receive Type
	pointer + 52 Frame Correlator Save
	pointer + 54 Beaconing station Upstream Node Address
	pointer + 60 Reserved
	pointer + 64 Beaconing station physical drop number
X'0A08'	Pointer to MAC buffer (a special buffer used by the software to transmit Adapter generated MAC frames)

READ ADAPTER BUFFER Completion.

Upon completion of the **READ ADAPTER BUFFER** command, the SSB will be loaded with **READ ADAPTER BUFFER** completion status as shown in Figure 3-40.

SSB ADDRESS	+ 0	X'000B'
	+ 2	READ COMPLETION

Figure 3-40. READ ADAPTER BUFFER Command SSB

Table 3-24 describes the READ ADAPTER BUFFER Completion field bit definitions.

Table 3-24. READ ADAPTER BUFFER Completion Field Bit Definitions

BIT 0	COMMAND COMPLETE. Bit 0 is set when the **READ ADAPTER BUFFER** command is completed.
BITS 1-15	RESERVED. Bits 1-15 are reset to zero.

Freeze-Dump

The Freeze-Dump state of the Adapter is entered by executing back-to-back resets followed by a **FREEZE** command issued to the Command/Status register. The reset may be executed by either writing to the Reset register or activating the RESET DRV line. The Freeze-Dump utility "freezes" the adapter in its current state and allows the use of commands not ordinarily recognized by the Adapter. Three commands are available in the Freeze-Dump state, the **STEP** command, the **WRITE** command, and the **EXECUTE** command. All of the Freeze-Dump commands are written to the Command/Status register.

FREEZE Command (X'80')

The **FREEZE** command is used to set the Adapter in the Freeze-Dump state.

STEP Command (X'FF')

The **STEP** command increments the ADDRESS Register by X'0800'. Normally, the ADDRESS Register is limited to a 2K-byte address range. The **STEP** command allows paging beyond the range normally allowed the Attached System. The ADDRESS Register is initialized after the system reset.

WRITE Command (X'E6')

The **WRITE** command allows the Host to write to the Adapter's memory using the DATA Auto/Inc Register.

EXECUTE Command (X'C5')

The **EXECUTE** command allows the execution of software code at a user specified location. The user specified location should be contained in the vector at X'0000'.

Chapter 4. Environmental Requirements

CONTENTS

Physical Requirements

Power Supply Requirements

The Adapter requires +5 volts (VCC) +/- 5% and ground supplied through the card I/O pins.

Current and Power

The current requirements and power dissipation for the Adapter are outlined below.

	Vcc(nom)	Icc(nom)	Icc(max)	Power(nom)	Power(max)
Note 1	5.00V	1.1A	1.3A	5.5W	6.5W
Note 2	5.00V	1.47A	1.53A	7.5W	8.25W

Notes:
1. For card assemblies 56X2295 and 69X8139.
2. For card assemblies other than 56X2295 or 69X8139.

Non-operating Requirements

The Adapter, when properly packaged in the shipping container, will withstand the following non-operating (storage) conditions:

Storage Air Temperature = 33-140°F (0.6-60°C)
Storage Relative Humidity = 5-80%

Operating Requirements

The Adapter will meet the minimal performance requirements while operating under the following environmental conditions:

Temperature +10 to +60°C
Relative Humidity 20-80%

These are generally defined as a building environment which is not air conditioned with normal winter heating and normal ventilation. The conditions stated are card ambient requirements.

Hot-Pluggability

The Adapter card is not hot-pluggable and therefore must be plugged into the RT PC before power is applied.

Appendix A. Software Example

Appendix A contains an example of software required to execute Adapter commands. This is intended as sample software and thus functionality and completeness are not claimed. The program is written in assembly language but comments are provided to assist the reader.

```
/*
 * 5799-CGZ (C) COPYRIGHT IBM CORPORATION 1986
 * LICENSED MATERIALS - PROPERTY OF IBM
 * REFER TO COPYRIGHT INSTRUCTIONS FORM NUMBER G120-2083
 */
/* $Header: if_lan.c,v 2.10 86/07/08 14:11:58 Exp $ */
/* $Source: /usr/sys/if_lan.c,v $ */

#ifndef lint
static char *rcsid = "$Header: if_lan.c,v 2.10 86/07/08 14:11:58 Exp $";
#endif

/*
 * IBM Token-Ring Local Area Network Adapter Driver
 */

#include "lan.h"
#if NLAN > 0

#include "../machine/pte.h"
#include "../h/param.h"
#include "../h/dir.h"
#include "../h/user.h"
#include "../h/mbuf.h"
#include "../h/buf.h"
#include "../h/protosw.h"
#include "../h/socket.h"
#include "../h/vmmac.h"
#include "../h/ioctl.h"
#include "../h/errno.h"
#include "../h/un.h"

#include "/usr/src/include/netdb.h"

#include "../net/if.h"
#include "../net/netisr.h"
#include "../net/route.h"
#include "../netinet/in.h"
#include "../netinet/in_systm.h"
#include "../netinet/ip.h"
#include "../netinet/ip_var.h"
#include "../netinet/if_ether.h"
#include "../netpup/pup.h"
```

```
#include "../machine/io.h"
#include "../machineio/ioccvar.h"
#include "../machineio/dmareg.h"
#include "../machine/debug.h"
#include "../h/kernel.h"

#include "../machineif/if_lanreg.h" /*appended at end of this listing*/
#include "../machineif/if_landma.h" /*appended at end of this listing*/
#include "../machineif/if_lanvar.h" /*appended at end of this listing*/
#include "../machineif/if_lanio.h"  /*appended at end of this listing*/

#ifdef DEBUG
char lan_debug = 0x00;                  /* controls printf's for debugging */
#endif DEBUG

int lan_probe(), lan_attach();
int lan_intr(), lan_init(), lan_ioctl(), lan_output(), lan_reset();
unsigned short mior();
struct mbuf *lan_get();

caddr_t lan_std[] = {
        (caddr_t)0xf00001c0, (caddr_t)0xf0000140, 0
};
struct iocc_device *lan_info[NLAN];

struct iocc_driver landriver = {
        lan_probe, 0, lan_attach, 0, lan_std, "lan", lan_info,
        0, 0, lan_intr, LAN_CMDREG
};

int lan_initialize(), lan_ring_watch(), lan_thaw();
struct lan_softc lan_softc[NLAN];
char lan_xid_resp[LAN_L_XID_RESP] = { '\201', '\001', '\002'};

/*
 * lan_probe - adapter does not interrupt in this state
 */
lan_probe(addr)
        register caddr_t addr;
{
        DEBUGF(lan_debug, printf("lan adapter probed\n");
        );
        return (PROBE_NOINT);
}


/*
 * lan_attach - make interface available to network software if
 *              autoconfig determines that interface exists.
 */

lan_attach(iod)
        register struct iocc_device *iod;
{
        register struct lan_softc *lns = &lan_softc[iod->iod_unit];
        register struct ifnet *ifp = &lns->lns_if;
        register struct sockaddr_in *sin;

        ifp->if_unit = iod->iod_unit;
        ifp->if_name = "lan";
        ifp->if_mtu = LAN_MTU;
        sin = (struct sockaddr_in *) & lns->lns_if.if_addr;
        sin->sin_family = AF_INET;
        sin->sin_addr = arpmyaddr((struct arpcom *)0);
        ifp->if_init = lan_init;
        ifp->if_ioctl = lan_ioctl;
        ifp->if_output = lan_output;
        ifp->if_reset = lan_reset;
        if_attach(ifp);

        DEBUGF(lan_debug, printf("lan%d attached\n", iod->iod_unit);
        );
}


/*
 * lan_reset - reset interface
 */

lan_reset(unit)
        register unsigned int unit;
{
        register struct iocc_device *iod;

        if (unit < NLAN && (iod = lan_info[unit]) != 0 && iod->iod_alive != 0)
                lan_init(unit);
        DEBUGF(lan_debug, printf("lan%d reset\n", unit);
        );
}


/*
 * lan_init - initialize and open adapter
 */

lan_init(unit)
        register int unit;
{
        register struct lan_softc *lns = &lan_softc[unit];
        struct ifnet *ifp = &lns->lns_if;
        struct sockaddr_in *sin;
        register struct lan_device *addr;

        sin = (struct sockaddr_in *) &ifp->if_addr;
        if (sin->sin_addr.s_addr == 0)
                return;

        DEBUGF(lan_debug, printf("lan%d begin init\n", unit);
        );

        addr = (struct lan_device *)(lan_info[unit]->iod_addr);

        if((lns->lns_if.if_flags & IFF_RUNNING) == 0) {
                /* read node address pointer once */
                lns->lns_adapter &= LAN_ADAP_BIA_READ;
                if (lan_initialize(unit) == 0) {
                        lns->lns_if.if_flags ]= IFF_RUNNING;
                        if (!(lns->lns_adapter & LAN_ADAP_BIA_READ)) {
                                lns->lns_ctl->lan_bia.adap_addr = LAN_ADDRESSES;
                                lns->lns_ctl->lan_bia.count = LAN_L_ADDR;
                                lns->lns_ctl->lan_bia.flag ]= LAN_ADDR_PENDING;
                                lan_exec(LAN_RDADAPTR, &lns->lns_ctl->lan_bia, addr, unit);
                        }
                        DEBUGF(lan_debug, printf("lan%d: end init\n", unit);
                        );

                } else {
                        lns->lns_adapter ]= LAN_ADAP_BROKEN;
                        lns->lns_if.if_flags &= ~(IFF_RUNNING ] IFF_UP);
                }
        }
        if(lns->lns_if.if_flags & IFF_RUNNING) {
                if((lns->lns_adapter & LAN_ADAP_OPEN) == 0) {
                        lan_open(unit, addr);  /* Open adapter */
                }
                if_rtinit(&lns->lns_if, RTF_UP);
                arpattach(&lns->lns_ac);
        }
}


 /* end lan_init */

/*
 * lan_start - start output to the adapter
 */

lan_start(lns, addr, xbuf, unit)
        register struct lan_softc *lns;
        register struct lan_device *addr;
        register int xbuf;
        register int unit;
{

        lns->lns_oactive = 1;
        lns->lns_xbuf = xbuf;
        DEBUGF(lan_debug,
                printf("lan%d start xmit buf %x length = %x\n",
                unit, xbuf, lns->xp[xbuf]->frame_size);
        );
        /* execute transmit command */
        lan_exec(LAN_TRANSMIT, lns->xp[xbuf], addr, unit);
}


/*
 * lan_ssb_clear - clear adapter -> system interrupt
 */

lan_ssb_clear(addr)
        register struct lan_device *addr;
{
        miow(&(addr->lan_cmdstat), LAN_SSBCLEAR);
}
```

```
/*
 * lan_recv - get mbufs and initialize receive lists for reception
 *              into mbuf data area
 */

lan_recv(unit, addr)
        register int unit;
        register struct lan_device *addr;
{
        register struct lan_softc *lns = &lan_softc[unit];
        register struct lan_recv_ctl *rcv;
        struct mbuf *m, *p;
        register int tcw;
        int i;
        /*
         * use an mbuf to hold
         * receive lists and headers
         */
        if(lns->lns_recv == 0) {
                MGET(m, M_DONTWAIT, MT_DATA);
                if (m == 0)
                        return(ENOBUFS);
                lns->lns_recv = mtod(m, struct lan_recv_ctl *);
        }
        rcv = lns->lns_recv;
        tcw = lan_dma_setup(rcv, TCW_RESERVE, lns->lan_dma_chan);
        for (i = 0; i < 2; i++) {
                lns->rp[i] = &rcv->lan_rlist[i];
                lns->rh[i] = &rcv->list_hdr[i];
                /* set up lan header areas */
                lns->rp[i]->d_parm[0].d_cnt =
                        (((short)(sizeof(struct list_hdr))) ] LAN_CHAIN);
                lns->rp[i]->d_parm[0].d_haddr = DMA_HI_ADDR(tcw);
                lns->rp[i]->d_parm[0].d_laddr = DMA_LO_ADDR(lns->rh[i], tcw);
                lns->rp[i]->cstat = LAN_R_CSTAT_REQ;
                lns->rp[i]->xlp_h =
                        DMA_HI_ADDR(tcw);
        }
        lns->rp[0]->xlp_l =
                DMA_LO_ADDR(lns->rp[1], tcw);
        lns->rp[1]->xlp_l =
                DMA_LO_ADDR(lns->rp[0], tcw);
        /*
         * use large mbufs for
         * receive buffers
         */
        for (i = 0; i < 2; i++) {
                if(lns->lns_rdata[i] == 0) {
                        MGET(m, M_DONTWAIT, MT_DATA);
                        if (m == 0)
                                return(ENOBUFS);
                        MCLGET(p, 1);
                        if (p == 0)
                                return(ENOBUFS);
                        else
                                m->m_off = (int)p - (int)m;
                        lns->lns_rbufp[i] = m;
                        lns->lns_rdata[i] = p;
                }
                tcw = lan_dma_setup(lns->lns_rdata[i],TCW_RESERVE, lns->lan_dma_chan);
                lns->rp[i]->d_parm[1].d_cnt = LAN_MTU & (~LAN_CHAIN);
                lns->rp[i]->d_parm[1].d_haddr = DMA_HI_ADDR(tcw);
                lns->rp[i]->d_parm[1].d_laddr = DMA_LO_ADDR(lns->lns_rdata[i], tcw);
        }
        return(0);
}


/*
 * lan_exec - serialize command requests to the adapter;
 *              do one at a time for now except for
 *              receive which needs to be outstanding.
 */
lan_exec(cmd, plist, addr, unit)
        register short cmd;
        register short *plist;
        register struct lan_device *addr;
        register int unit;
{
        struct lan_softc *lns = &lan_softc[unit];
        register struct lan_ctl *ctl = lns->lns_ctl;
        register int i;
        int tcw;
        int s = splimp();

        tcw = lan_dma_setup(plist, TCW_RESERVE, lns->lan_dma_chan);
        for (i = 5*ONESEC; i>0; i--) {
                DELAY(1);
                if (ctl->lan_scb.command == 0)  /* wait until scb cleared */
                        break;
        }
        if(i>0) {
                ctl->lan_scb.command = cmd;
                ctl->lan_scb.h_addr = DMA_HI_ADDR(tcw);
                ctl->lan_scb.l_addr = DMA_LO_ADDR(plist, tcw);
                DEBUGF(lan_debug,
                        printf("lan%d exec cmd %x, addr %x %x\n",
                                unit, ctl->lan_scb.command,
                                ctl->lan_scb.h_addr, ctl->lan_scb.l_addr);
                );
                miow(&(addr->lan_cmdstat), LAN_EXECUTE);
        } else printf("lan%d: adapter jammed\n",unit);
        splx(s);
}


/*
 * lan_open - open adapter
 */

lan_open(unit, addr)
        register int unit;
        register struct lan_device *addr;
{
        register struct lan_softc *lns = &lan_softc[unit];
        register struct lan_ctl *ctl = lns->lns_ctl;
        if((lns->lns_adapter & LAN_OPEN_IN_PROGRESS) == 0) {
                lns->lns_adapter |= LAN_OPEN_IN_PROGRESS;
                ctl->open_parm[0] = LAN_OPEN_OPTIONS;
                ctl->open_parm[1] = ctl->open_parm[2] = ctl->open_parm[3] = 0;
                ctl->open_parm[4] = ctl->open_parm[5] =
                        ctl->open_parm[6] = ctl->open_parm[7] = 0;
                ctl->open_parm[8] = LAN_OPEN_RLIST;
                ctl->open_parm[9] = LAN_OPEN_XLIST;
                ctl->open_parm[10] = LAN_OPEN_BUFSIZE;
                ctl->open_parm[11] = LAN_OPEN_RAMSTART;
                ctl->open_parm[12] = LAN_OPEN_RAMEND;
                ctl->open_parm[13] = LAN_OPEN_XMINMAX;

                lan_exec(LAN_OPEN, ctl->open_parm, addr, unit);
                DEBUGF(lan_debug,
                        printf("lan%d attempting to open addr %x %x %x\n",
                                unit, ctl->open_parm[1],
                                ctl->open_parm[2], ctl->open_parm[3]);
                );
        }
}


/*
 * lan_intr - adapter interrupt handler
 */

lan_intr(unit)
        register int unit;
{
        register struct lan_softc *lns = &lan_softc[unit];
        register struct lan_device *addr = (struct lan_device *)lan_info[unit]->iod_addr;
        struct lan_ctl *ctl = lns->lns_ctl;
        register struct mbuf *m;
        register short sifrbuf, sifwbuf;
        struct ifqueue *inq;
        struct mbuf *p;
        char type, *c;
        unsigned short len;
        register int next_buf;
        int k, i, tcw;
        short retparm[4];
        short *retptr;
        unsigned short *initptr, *shortptr;
        unsigned short open_phase, open_status;
        struct ifnet *ifp = &lns->lns_if;
        struct sockaddr_in *sin;
        int alt;

        sifrbuf = (mior(&(addr->lan_cmdstat)));
        if ((sifrbuf & LAN_INT) == 0) {
                return (1);
        }
        sifrbuf &= LAN_ADAP_INT;        /* interrupt type */
        DEBUGF(lan_debug,
                printf("lan%d int: status = 0x%b, code = 0x%x, ssb = %x,%x,%x,%x\n",
                        unit, sifrbuf & 0xfff0, LAN_STAT_BITS, sifrbuf & 0x000f,
                        ctl->lan_ssb.command, ctl->lan_ssb.status0,
                        ctl->lan_ssb.status1, ctl->lan_ssb.status2);
        );
        switch (sifrbuf) {
        case LAN_RECVSTAT:
                /*
                 * Receive notify
                 *
                 * When frame complete, copy data to mbuf chain.
                 * Signal adapter to continue.  Then either enqueue IP
                 * packets or process arp packets.
                 */

                /* determine buffer(s) from receive list address */
                if ((ctl->lan_ssb.status1 == lns->rp[1]->xlp_h) &&
                        (ctl->lan_ssb.status2 == lns->rp[1]->xlp_l))
                        i = 0;
                        else i = 1;
                if (ctl->lan_ssb.status0 == LAN_FRAME_COMPLETE) {
                        DEBUGF(lan_debug,
                                printf("lan%d recv: status=0x%b, cstat=0x%b, length=%x\n",
                                        unit, ctl->lan_ssb.status0, LAN_RECV_BITS,
                                        lns->rp[i]->cstat, LAN_RCSTAT_BITS,
                                        lns->rp[i]->frame_size);
                        );
                lan_ssb_clear(addr);
                alt = 0;
                /* if alternate buffer full, get that one first */
                if (lns->rp[1-i]->cstat & LAN_RCSTAT_COMPLETE) {
                        alt=LAN_RECV_ALT;
                        i=1-i;
                        }
                check_alt:
                        lns->lns_if.if_ipackets++;
                        len = lns->rp[i]->frame_size - sizeof(struct list_hdr);
                        if (len == 0)
                                goto chk_xid;
                        p = lns->lns_rdata[i];
                        m = lan_get(p, len);
                        type = lns->rh[i]->dsap;
                        if (m == 0)
                                goto clear;
                chk_xid:
                        if((lan_xid_test(m, i, unit)) ]] (len == 0))
                                goto clear;
                        switch (type) {
#ifdef INET
                        case LAN_IPTYPE:
                                schednetisr(NETISR_IP);
                                inq = &ipintrq;
                                break;
                        case LAN_ARPTYPE:
                                arpinput(&lns->lns_ac, m);
                                goto clear;
#endif
                        default:
                                DEBUGF(lan_debug,
                                        printf("lan%d packet not recognized, freeing mbuf\n",
                                                unit);
                                );
                                m_freem(m);
                        goto clear;
                }
                if (IF_QFULL(inq)) {
                        DEBUGF(lan_debug, printf("lan%d: ip qfull\n", unit);
                        );
                        IF_DROP(inq);
                        m_freem(m);
                        goto clear;
                }
                IF_ENQUEUE(inq, m);

        }
        else lan_ssb_clear(addr);
        clear:
        lns->rp[i]->cstat = LAN_R_CSTAT_REQ;
        miow(&(addr->lan_cmdstat), LAN_RECVALID);
        if (alt) {
                alt = 0;
                i=1-i;
                goto check_alt;
                }
        break;

case LAN_XMITSTAT:
        /*
         * Transmit status
         *
         * If next buffer full, start output.
         * Dequeue next IP buffer.
         */

        lan_ssb_clear(addr);
        lns->lns_if.if_opackets++;

        if ((ctl->lan_ssb.status1 == lns->xp[1]->xlp_h) &&
                (ctl->lan_ssb.status2 == (lns->xp[1]->xlp_l & ~LAN_ODD_PTR)))
                next_buf = 1;
                else next_buf = 0;
        if (lns->xp[next_buf]->cstat & LAN_XMIT_VALID) {
                lan_start(lns, addr, next_buf, unit);
                if (lns->lns_if.if_snd.ifq_head) {
                        struct mbuf *m;
                        IF_DEQUEUE(&lns->lns_if.if_snd, m);
                        if (lan_put(lns, m, 1 - next_buf, unit) != 0)
                                m_freem(m);
                }
        } else
                lns->lns_oactive = 0;
        break;

case LAN_ACHECK:
        /*
         * Adapter check - Retry adapter initialization
         */
        lns->lns_adapter ]= LAN_ADAP_BROKEN;
        lns->lns_if.if_flags &= ~(IFF_RUNNING ] IFF_UP);
        sifwbuf = LAN_ACHECK_DATA;
        shortptr = &(addr->lan_address);
        initptr = &(addr->lan_data);
        for (k = ADAP_CHK_SIZE, retptr = ((short *)retparm); k > 0; k--, retptr++) {
                miow(shortptr, sifwbuf);
                *retptr = mior(initptr);
                sifwbuf = sifwbuf + 2;
        }
        printf("lan%d: unrecoverable token ring adapter failure, ", unit);
        printf("adapter check field = 0x%b, parm0=0x%x, parm1=0x%x, parm2=0x%x\n",
                retparm[0], LAN_ACHECK_BITS, retparm[1], retparm[2], retparm[3])
        if (lns->lns_freezer != 0) {
                lan_freeze(unit); /* freeze the adapter */
                wakeup(lns);
                timeout(lan_thaw, (caddr_t)unit, 180 * hz);
        } else {
                lan_ssb_clear(addr);
                timeout(lan_reset, (caddr_t)unit, 1 * hz);
        }
        if (retparm[0] & (XMIT_PARITY ]] XMIT_UNDERRUN)) {
                lns->lns_if.if_oerrors++;
        }
        if (retparm[0] & (RECV_PARITY ]] RECV_OVERRUN)) {
                lns->lns_if.if_ierrors++;
        }
        break;

case LAN_RINGSTAT:
        /*
         * Ring status
         *
         * Processing depends on ring status.  For
         * ring error set timer to check again in 60
         * seconds to allow for possible ring
         * recovery.  Certain statuses update adapter
         * state.  Others informational only.
         */

        DEBUGF(lan_debug,
                printf("lan%d ring status = %b\n",
                        unit, ctl->lan_ssb.status0, LAN_RING_BITS);
        );
        if (ctl->lan_ssb.status0 & LAN_AUTOER1) {
                lns->lns_adapter ]= LAN_ADAP_AUTOER1;
                lns->lns_if.if_flags &= ~IFF_UP;
                lan_ssb_clear(addr);
                lan_open(unit, addr);
                break;
        } else if (ctl->lan_ssb.status0 & LAN_REMOVE_RECV) {
                lns->lns_adapter &= ~LAN_ADAP_OPEN;
                lns->lns_adapter ]= LAN_ADAP_DOWN;
                lns->lns_if.if_flags &= ~IFF_UP;
                printf("lan%d: removed from network\n", unit);
        } else if (ctl->lan_ssb.status0 & LAN_WIRE_FAULT) {
                printf("lan%d: cable failure\n", unit);
                lan_ssb_clear(addr);
                lns->lns_adapter &= ~LAN_ADAP_OPEN;
                lns->lns_if.if_flags &= ~IFF_UP;
                lns->lns_ring ]= LAN_CABLE_FAIL;
                if(!lns->lns_ring_watch) {
                        lns->lns_ring_watch++;
                        timeout(lan_ring_watch, (caddr_t)unit, SIXTY * hz);
                }
                break;
        } else if ( ctl->lan_ssb.status0 &
                (LAN_SIGNAL_LOSS | LAN_HARD_ERROR | LAN_XMIT_BEACON)
                ) {
                if (lns->lns_ring & LAN_RECOVERY)
                        lns->lns_ring = ctl->lan_ssb.status0 | LAN_RECOVERY;
                else {
                        lns->lns_ring |= LAN_RECOVERY;
                        if (ctl->lan_ssb.status0 & LAN_XMIT_BEACON)
                                printf("lan%d: beaconing\n", unit);
                        if(!lns->lns_ring_watch) {
                                lns->lns_ring_watch++;
                                timeout(lan_ring_watch, (caddr_t)unit, SIXTY * hz);
                        }
                }
        }
        lns->lns_ring &= ~LAN_RECOVERY;
        if (ctl->lan_ssb.status0 & LAN_SINGLE) {
                printf("lan%d: single station on network\n", unit);
        }
        lan_ssb_clear(addr);
        break;


case LAN_CMDSTAT:
        /*
         * Command status
         *
         * On open, examine completion status.  Set
         * adapter state and, if failure, print
         * informational message and retry open if error
         * not permanent.  If success, set up transmit
         * lists and begin output.
         */

        DEBUGF(lan_debug, printf("lan%d cmd stat: %x %x %x %x\n",
                unit, ctl->lan_ssb.command, ctl->lan_ssb.status0,
                ctl->lan_ssb.status1, ctl->lan_ssb.status2);
        );
        if (ctl->lan_ssb.command == LAN_OPEN) {
                if (ctl->lan_ssb.status0 == LAN_OPEN_COMPLETE) {
                        lan_ssb_clear(addr);
                        DEBUGF(lan_debug, printf("lan%d open complete\n", unit);
                        );
                        lns->lns_adapter &= LAN_ADAP_BIA_READ;
                        lns->lns_adapter |= LAN_ADAP_OPEN;
                        lns->lns_ring &= ~(
                                LAN_RECOVERY |
                                LAN_XMIT_BEACON |
                                LAN_CABLE_FAIL);
                        ctl->lan_bia.adap_addr = lns->lns_node_addr;
                        ctl->lan_bia.count = LAN_L_ADDR;
                        lan_exec(LAN_RDADAPTR, &ctl->lan_bia, addr, unit);
                } else {
                        lns->lns_adapter &=
                                ~(LAN_ADAP_OPEN | LAN_OPEN_IN_PROGRESS);
                        /* adapter open error */
                        lns->lns_if.if_flags &= ~IFF_UP;
```

```
DEBUGF(lan_debug,
        printf("lan%d open failure, status = 0x%b, error = 0x%x\n",
                unit, ctl->lan_ssb.status0 & 0xff00,
                LAN_OPEN_STAT_BITS,
                ctl->lan_ssb.status0 & 0x00ff);
);
open_phase =
        ctl->lan_ssb.status0 & ~LAN_OPEN_PHASE_MASK;
open_status =
        ctl->lan_ssb.status0 & LAN_OPEN_PHASE_MASK;
lan_ssb_clear(addr);
if (open_status & LAN_OPEN_NODE_ERROR) {
        printf("lan%d: node address error\n", unit);
        lns->lns_adapter |= LAN_ADAP_DOWN;
        break;
} else if (lns->lns_adapter & LAN_ADAP_AUTOER1) {
        lns->lns_adapter &= ~LAN_ADAP_AUTOER1;
        printf("lan%d: hardware error, ", unit);
        printf("adapter removed from ring\n");
        break;
}
        printf("lan%d: open err=0x%x",unit, open_status);
        printf("%s-n", open_errmsg[open_status & 0x0f]);
        switch (open_status) {
        case LAN_OPEN_TIMEOUT:
        case LAN_OPEN_OREQ_PARM:
        case LAN_OPEN_OIMPL:
                if(lns->lns_open_retries < LAN_MAX_OPEN_RETRY)
                {
                        lns->lns_open_retries++;
                        lan_open(unit,addr);
                }
                else lns->lns_open_retries = 0;
                break;
        case LAN_OPEN_FUNC_FAILURE:
                if ((open_phase == LAN_OPEN_LOBE_TEST) ||
                (open_phase == LAN_OPEN_INSERTION))
                {
                        lns->lns_ring |= LAN_CABLE_FAIL;
                        if(!lns->lns_ring_watch) {
                                lns->lns_ring_watch++;
                                timeout (lan_ring_watch,
                                (caddr_t)unit, SIXTY * hz);
                        }
                }
                else
                if(lns->lns_adapter & LAN_ADAP_FCTNFAIL == 0)
                {
                        lns->lns_adapter |= LAN_ADAP_FCTNFAIL;
                        lan_open(unit,addr);
                }
                break;
        case LAN_OPEN_OSIGNAL_LOSS:
        case LAN_OPEN_RING_FAILURE:
        case LAN_OPEN_RING_BEACON:
                if(!lns->lns_ring_watch) {
                        lns->lns_ring_watch++;
                        timeout(lan_ring_watch, (caddr_t)unit, SIXTY * hz);
                }
```

```
                        }
        }
} else if (ctl->lan_ssb.command == LAN_RDADAPTR) {
        lan_ssb_clear(addr);
        if (ctl->lan_bia.flag & LAN_ADDR_PENDING) {
                lns->lns_node_addr = ctl->lan_bia.count;
                ctl->lan_bia.flag &= ~LAN_ADDR_PENDING;
        } else {
                bcopy(&ctl->lan_bia.count, lns->lns_addr, LAN_L_ADDR);
                printf ("lan%d: address ",unit);
                c = (char *)lns->lns_addr;
                for (i = 0; i < LAN_L_ADDR; i++) {
                        if (i != 0) printf (":");
                        printf ("%x", *c++);
                }
                printf("\n");
                /* allocate transmit list buffers very first time */
                if ((lns->lns_adapter & LAN_ADAP_BIA_READ) == 0) {
                        /* allocate xmit lists */
                        for (i = 0; i < LAN_XMITLIST_CT; i++) {
                                MGET(m, M_DONTWAIT, MT_DATA);
                                if (m == 0)
                                        goto lan_mbuf_fail;
                                MCLGET(p, 1);
                                if (p == 0)
                                        goto lan_mbuf_fail;
                                else
                                        m->m_off = (int)p - (int)m;
                                lns->xp[i] = (struct lan_list *)p;
                                lns->lns_xbufp[i] = (char *) ((int)lns->xp[i]
                                + sizeof(struct lan_list));
                                tcw = lan_dma_setup(lns->lns_xbufp[i],TCW_RESERVE,
                                lns->lan_dma_chan);
                                lns->xp[i]->d_parm[0].d_haddr = DMA_HI_ADDR(tcw);
                                lns->xp[i]->d_parm[0].d_laddr =
                                DMA_LO_ADDR(lns->lns_xbufp[i], tcw);
                        }
                        lns->lns_oactive = 0;
                        lns->lns_adapter ]= LAN_ADAP_BIA_READ;
                }
                for (i = 0; i < LAN_XMITLIST_CT; i++) {
                        tcw = lan_dma_setup(lns->xp[i],TCW_RESERVE,lns->lan_dma_chan);
                        lns->xp[1-i]->xlp_h =
                                DMA_HI_ADDR(tcw);
                        lns->xp[1-i]->xlp_l =
                                (DMA_LO_ADDR(lns->xp[i], tcw)) ] LAN_ODD_PTR;
                        lns->xp[i]->cstat = LAN_XCSTAT_COMPLETE;
                }
                /* reset receive lists */
                if (lan_recv(unit, addr) != 0) goto lan_mbuf_fail;
                lan_exec(LAN_RECEIVE, lns->rp[0], addr, unit);
                lns->lns_if.if_flags ]= IFF_UP;
                sin = (struct sockaddr_in *) & ifp->if_addr;
                arpwhohas(&lns->lns_ac, &sin->sin_addr);
                if (lns->lns_if.if_snd.ifq_head) {
                        struct mbuf *m;
```

```
                                        IF_DEQUEUE(&lns->lns_if.if_snd, m);
                                        if (lan_put(lns, m, 0, unit) == 0) {
                                                lan_start(lns, addr, 0, unit);
                                                if (lns->lns_if.if_snd.ifq_head) {
                                                        IF_DEQUEUE(&lns->lns_if.if_snd,
                                                                m);
                                                        if (lan_put(lns, m, 1, unit) != 0)
                                                                m_freem(m);
                                                }
                                        } else
                                                m_freem(m);
                                }
                        }
                } else if (ctl->lan_ssb.command == LAN_CLOSE) {
                        if (ctl->lan_ssb.status0 == LAN_CLOSE_COMPLETE) {
                                lns->lns_adapter ]= LAN_ADAP_DOWN;
                                DEBUGF(lan_debug, printf("lan%d closed\n", unit);
                                );
                        }
                        lan_ssb_clear(addr);
                } else {
                        if (ctl->lan_ssb.command == LAN_SSB_REJECT)
                                printf("lan%d: command reject, internal software error, ",
                                        unit);
                                printf("reject reason = 0x%b, command = 0x%x.\n",
                                        ctl->lan_ssb.status0, LAN_CMD_REJ_BITS,
                                        ctl->lan_ssb.status1);
                        lan_ssb_clear(addr);
                }
                break;
lan_mbuf_fail:
                printf("lan%d: mbufs.\n", unit);
                break;


        default:
                DEBUGF(lan_debug,
                        printf("lan%d unexpected interrupt: status = 0x%b, code = 0x%x\n",
                                unit, sifrbuf & 0xfff0, LAN_STAT_BITS, sifrbuf & 0x000f);
                );
                lan_ssb_clear(addr);
                break;
        }
        /* end switch */

        *LAN_IRQ12 = 0;                 /* enable interrupt level */
        return (0);
}


/*
 * lan_ioctl - adapter ioctl
 */
lan_ioctl(ifp, cmd, data)
        register struct ifnet *ifp;
        register int cmd;
        register caddr_t data;
{
        register struct ifreq *ifr = (struct ifreq *)data;
register int s = splimp();
register int error = 0;
register struct lan_softc *lns = &lan_softc[ifp->if_unit];
struct lan_device *addr = (struct lan_device *)
lan_info[ifp->if_unit]->iod_addr;
unsigned short *dump, dump_data;
int i, j;

switch (cmd) {
case SIOCSIFADDR:
        if (ifp->if_flags & IFF_RUNNING)
                if_rtinit(ifp, -1); /* delete previous route */
        lan_setaddr(ifp, (struct sockaddr_in *) & ifr->ifr_addr);
        lan_init(ifp->if_unit);
        break;
case SIOCSIFFLAGS:
        if((ifr->ifr_flags & IFF_UP)
                && ((lns->lns_adapter & LAN_ADAP_OPEN) == 0)) {
                lan_init(ifp->if_unit);
        } else if (((ifr->ifr_flags & IFF_UP) == 0)
                && (lns->lns_adapter & LAN_ADAP_OPEN)) {
                lan_close(ifp->if_unit);
                }
        break;
case SIOCSLANDUMP:
        lns->lns_freezer = u.u_procp;
        sleep(lns,PZERO+1);
        break;
case SIOCFLANDUMP:
        lan_freeze(ifp->if_unit); /* freeze the adapter */
        timeout(lan_thaw, (caddr_t)ifp->if_unit, 180 * hz);
        break;
case SIOCGLANDUMP:
        if (lns->lns_adapter & LAN_ADAP_FROZEN) {
                dump = ((struct lan_dump *)ifr->ifr_data)->lan_dump_data;
                for (j = 0; j < LAN_FREEZE_DUMP / LAN_FREEZE_CHUNK; j++) {
                        if (j) {
                                miow(&(addr->lan_cmdstat), LAN_FREEZE_INCR);
                                DELAY(LAN_ADAP_MIN_RESET);
                        }
                        miow(&(addr->lan_address), 0);
                        for (i = 0; i < LAN_FREEZE_CHUNK / 2; i++, dump++) {
                                dump_data = mior(&(addr->lan_datai));
                                if(copyout( (caddr_t)&dump_data,
                                         (caddr_t)dump,sizeof(short)) !=0) {
                                        goto dump_done;
                                }
                        }
                }
        dump_done:
                lan_unfreeze(ifp->if_unit);
                timeout(lan_reset, (caddr_t)(ifp->if_unit), 1 * hz);
        }
        break;
default:
        error = EINVAL;
}
splx(s);
```

```
        return (error);
}


/*
 * lan_output - token ring output routine
 */
lan_output(ifp, m0, dst)
        register struct ifnet *ifp;
        register struct mbuf *m0;
        struct sockaddr *dst;
{
        int error;
        char type;
        u_char edst[LAN_L_ADDR];
        struct in_addr idst;
        register struct lan_softc *lns = &lan_softc[ifp->if_unit];
        register struct mbuf *m = m0;
        register struct ether_header *un;
        extern struct ifnet loif;

        DEBUGF(lan_debug, printf("lan%d output request\n", ifp->if_unit);
        );

        switch (dst->sa_family) {
#ifdef INET
        case AF_INET:
                idst = ((struct sockaddr_in *)dst)->sin_addr;
                if (!arpresolve(&lns->lns_ac, m, &idst, edst))
                        return (0);
                type = LAN_IPTYPE;
                break;
#endif
        case AF_UNSPEC:
                un = (struct ether_header *)dst->sa_data;
                bcopy((caddr_t)un->ether_dhost, (caddr_t)edst, sizeof(edst));
                type = LAN_ARPTYPE;
                break;
        default:
                printf("lan%d: can't handle af%d.\n", ifp->if_unit, dst->sa_family);
                error = EAFNOSUPPORT;
                goto bad;
        }
        return (lan_output_llc(ifp, m0, edst, LAN_UI_CMD, type));
bad:
        m_freem(m0);
        return (error);
}


lan_output_llc(ifp, m0, dst, llc_ctl, type)
        register struct ifnet *ifp;
        register struct mbuf *m0;
        u_char * dst;
        char llc_ctl;
        char type;
{
        int s, error;
        u_char edst[LAN_L_ADDR];
```

```
struct lan_device *addr = (struct lan_device *)
lan_info[ifp->if_unit]->iod_addr;
register struct lan_softc *lns = &lan_softc[ifp->if_unit];
register struct mbuf *m = m0;
register struct list_hdr *lan;
extern struct ifnet loif;
/* Add token-ring header */

m = m_get(M_DONTWAIT, MT_HEADER);
if (m == 0) {
        error = ENOBUFS;
        goto bad;
}
m->m_next = m0;
m->m_off = MMINOFF;
m->m_len = sizeof(struct list_hdr);
lan = mtod(m, struct list_hdr *);
bcopy((caddr_t)dst, (caddr_t)lan->to_addr, sizeof(edst));
bcopy((caddr_t)lns->lns_addr, (caddr_t)lan->from_addr, LAN_L_ADDR);
lan->dsap = type;
lan->ssap = type;
if(llc_ctl != LAN_UI_CMD)
        lan->ssap ]= 0x01;
lan->llc_ctl = llc_ctl;
lan->pcf0 = LAN_PCF0;
lan->pcf1 = LAN_PCF1;
s = splimp();
if (IF_QFULL(&ifp->if_snd)) {
        IF_DROP(&ifp->if_snd);
        error = ENOBUFS;
        goto qfull;
}
if (!(lns->lns_adapter & (LAN_ADAP_BROKEN ] LAN_ADAP_DOWN))) {
        if (lns->lns_adapter & LAN_ADAP_OPEN) {
                int next_buf=lns->lns_nextbuf;
                if(lns->xp[next_buf]->cstat & LAN_XCSTAT_COMPLETE) {
                        if (lan_put(lns, m, next_buf, ifp->if_unit) == 0) {
                                DEBUGF(lan_debug,
                                        printf("lan%d xmit buf %x filled\n",
                                                ifp->if_unit, next_buf);
                                );
                                if (lns->lns_oactive == 0)
                                        lan_start(lns, addr, next_buf, ifp->if_unit);
                        } else
                                m_freem(m);
                } else
                        IF_ENQUEUE(&ifp->if_snd, m);
        } else {
                if ((lns->lns_adapter & LAN_OPEN_IN_PROGRESS) == 0) {
                        lan_open(ifp->if_unit, addr);
                }
                IF_ENQUEUE(&ifp->if_snd, m);
        }
} else
        m_freem(m);

splx(s);
return (0);
```

```
qfull:
        m0 = m;
        splx(s);
        DEBUGF(lan_debug, printf("lan%d IP output queue full\n", ifp->if_unit);
        );
bad:
        m_freem(m0);
        return (error);
}


/*
 * lan_ring_watch
 *
 * This routine is entered upon expiration of the 60-second interval
 * timer set to examine ring state and print status message.
 * If necessary, attempt to reopen the adapter.
 */

lan_ring_watch(unit)
        register int unit;
{
        register struct lan_device *addr = (struct lan_device *)lan_info[unit]->iod_addr;
        register struct lan_softc *lns = &lan_softc[unit];
        int s;

        if (lns->lns_ring & LAN_RECOVERY) {
                if (lns->lns_ring & LAN_XMIT_BEACON)
                        printf("lan%d: beaconing\n", unit);
        }
        s = splimp();
        if ((lns->lns_adapter & (
                LAN_OPEN_IN_PROGRESS ]
                LAN_ADAP_OPEN ]
                LAN_ADAP_BROKEN ]
                LAN_ADAP_DOWN
        )) == 0) {
                lan_open(unit, addr);
                timeout(lan_ring_watch, (caddr_t)unit, SIXTY * hz);
        }
        else lns->lns_ring_watch = 0;
        splx(s);
}

/*
 * lan_put - copy mbufs into large xmit.mbuf already mapped for dma operation
 */

lan_put(lns, m, xbuf, unit)
        struct lan_softc *lns;
        struct mbuf *m;
        int xbuf;
        int unit;
{
        struct lan_device *addr = (struct lan_device *)lan_info[unit]->iod_addr;
        struct mbuf *mp;
        register struct lan_list *xcp;
        register int total_length = 0;
        int s;
        register char *bp;

        xcp = lns->xp[xbuf];
        xcp->xlp_l ]= LAN_ODD_PTR;

        bp = ((char *) lns->lns_xbufp[xbuf]);
        for (mp = m; mp; mp = mp->m_next) {
                register int len = mp->m_len;

                bcopy(mtod(mp, char *), bp, len);
                bp += len;
                total_length += len;
        }
        xcp->frame_size = total_length;
        xcp->d_parm[0].d_cnt = total_length & (~LAN_CHAIN);
        lns->lns_nextbuf = 1-xbuf;
        s = splimp();
        xcp->cstat = LAN_X_CSTAT_REQ;
        if(lns->lns_oactive) {
                lns->xp[1-xbuf]->xlp_l &= ~LAN_ODD_PTR;
                miow(&(addr->lan_cmdstat), LAN_XMTVALID);
        }
        splx(s);
        m_freem(m);
        return (0);
}

/*
 * lan_get - copy from driver receive buffers into mbuf's
 */
struct mbuf *lan_get(faddr, totlen)
        u_char * faddr;
        register unsigned short totlen;
{
        register struct mbuf *m;
        struct mbuf *top = 0;
        register struct mbuf **mp = &top;
        register unsigned short len;
        register u_char * cp;

        cp = faddr;
        while (totlen > 0) {
                MGET(m, M_DONTWAIT, MT_DATA);
                if (m == 0)
                        goto bad;
                len = totlen;
                if (len >= CLBYTES) {
                        register struct mbuf *p;

                        MCLGET(p, 1);
                        if (p != 0) {
                                m->m_len = len = CLBYTES;
                                m->m_off = (int)p - (int)m;
                        } else {
                                m->m_len = len = MIN(MLEN, len);
                                m->m_off = MMINOFF;
                        }
                } else {
                        m->m_len = len = MIN(MLEN, len);
                        m->m_off = MMINOFF;
                }
                bcopy(cp, mtod(m, char *), len);
                cp += len;
                *mp = m;
                mp = &m->m_next;
                totlen -= len;
        }
        return (top);

bad:

        DEBUGF(lan_debug, printf("lan mbuf request failed\n");
        );
        if (top != 0)
                m_freem(top);
        return (0);
}


/*
 * lan_initialize - initialize adapter;
 *
 * Initialization consists of 3 phases:
 *      1) check of bring-up diagnostics
 *      2) transfer of initialization parameters
 *      3) dma interface check.
 */

lan_initialize(unit)
        register int unit;
{
        register struct lan_softc *lns = &lan_softc[unit];
        register struct lan_device *addr;
        int retry, success, failure, j, k; /* loop controls */
        unsigned short initparm[LAN_NUM_IPARMS];
        register unsigned short *parmptr, *initptr;
        unsigned short sifrbuf, sifwbuf;
        register unsigned short *shortptr;
        int tcw = 0;
        struct mbuf *m;
        char c;

        /*
         * assume initial attempt plus
         * 3 more retries of full procedure
         */

        DEBUGF(lan_debug, printf("lan%d initialization\n", unit);
        );
        addr = (struct lan_device *)(lan_info[unit]->iod_addr);
        /* disable dma channel */
        if ((caddr_t)addr == lan_std[0]) {
                lns->lan_dma_chan = DM_CHAN5;
                *((char *)CTL2_SMASK) = (CH_DISABLE ] CH5);
        } else {
                lns->lan_dma_chan = DM_CHAN6;
                *((char *)CTL2_SMASK) = (CH_DISABLE ] CH6);
        }
```

```
lan_dma_init(lns->lan_dma_chan);

/* begin adapter initialization */
for (retry = LAN_RETRY; retry > 0; retry--) {
        shortptr = &(addr->lan_cmdstat);
        miow(shortptr, LAN_RESET);
        /* check bring-up diagnostics results */
        DEBUGF(lan_debug,
                printf("lan%d init retry %d\n", unit, LAN_RETRY - retry);
        );
        for (j = LAN_RESET_WAIT, success = failure = 0; j > 0; j--) {
                DELAY(TEN_MS);
                sifrbuf = mior(shortptr);
                DEBUGF(lan_debug,
                        printf("lan%d status = 0x%b, int/err code = 0x%x\n",
                                unit, sifrbuf & 0xfff0,
                                LAN_STAT_BITS, sifrbuf & 0x000f);
                );
                if (sifrbuf & LAN_INITIALIZE) {
                        if ((sifrbuf &
                                (LAN_TEST ]
                                LAN_ERROR ]
                                LAN_ADAP_INT)) == 0) {
                                /* diagnostics successful */
                                success++;
                                break;
                } else if ((sifrbuf & LAN_TEST) &&
                                (sifrbuf & LAN_ERROR)) {
                                /* unrecoverable error */
                                failure++;
                                break;
                        }
                }
        }
        DEBUGF(lan_debug, printf("lan%d pods complete\n", unit);
        );                      /* end bring-up diagnostics */

        if (success) {          /* diagnostics ok,now handshake */
                /* transfer initialization parms */
                *LAN_IRQ12 = 0; /* enable interrupt level */
                success = 0;
                sifrbuf = mior(&(addr->lan_enable));
                /*
                 * use an mbuf to store scb, ssb
                 * to guarantee alignment
                 */
                if(lns->lns_ctl == 0) {
                        MGET(m, M_DONTWAIT, MT_DATA);
                        if (m == 0) {
                                failure++;
                                break;
                        }
                        lns->lns_ctl = mtod(m, struct lan_ctl *);
                }
                tcw = lan_dma_setup(lns->lns_ctl, TCW_RESERVE, lns->lan_dma_chan);
                initparm[0] = LAN_INIT_OPTIONS;
                initparm[1] = initparm[2] = initparm[3] = 0;
                initparm[4] = LAN_INIT_RBURST;
                initparm[5] = LAN_INIT_XBURST;
        initparm[6] = LAN_INIT_ABORT;
        initparm[7] = DMA_HI_ADDR(tcw);
        initparm[8] = DMA_LO_ADDR(lns->lns_ctl, tcw);
        initparm[9] = initparm[7];
        initparm[10] = DMA_LO_ADDR(&lns->lns_ctl->lan_ssb, tcw);
        /* write adapter init parameters */
        miow(&(addr->lan_address), LAN_INIT_DATAA);
        initptr = &(addr->lan_datai);
        for (k = LAN_NUM_IPARMS, parmptr = initparm;
                k > 0;
                k--, parmptr++) {
                miow(initptr, *parmptr);
        }
        /* read status to clear */
        c = *((char *)CTL2_CMD);
        if (lns->lan_dma_chan == DM_CHAN5) {
                /* set page mode for dma */
                *((char *)DMRA) &= ~CH5_PAGE;
                *((char *)CTL2_SMASK) = (CH_ENABLE ] CH5);
                *((char *)CTL2_MODE) = (DM_CASCADE ] CH5);
        } else {
                /* set page mode for dma */
                *((char *)DMRA) &= ~CH6_PAGE;
                *((char *)CTL2_SMASK) = (CH_ENABLE ] CH6);
                *((char *)CTL2_MODE) = (DM_CASCADE ] CH6);
        }
        miow(&(addr->lan_cmdstat), LAN_EXECUTE);

        /*
         * wait at least 10 seconds before detecting
         * initialization error to allow for dma timeout
         */

        for (k = LAN_DMA_TIMEOUT + 1; k > 0; k--) {
                DELAY(TEN_MS);
                sifrbuf = mior(&(addr->lan_cmdstat));
                if ((sifrbuf &
                        (LAN_INITIALIZE ]
                        LAN_TEST ]
                        LAN_ERROR)) == 0) {
                        DEBUGF(lan_debug,
                                printf("lan%d dma xface test ok\n",
                                        unit);
                        );
                        success++;
                        break;
                } else if (sifrbuf & LAN_ERROR) {
                        DEBUGF(lan_debug,
                                printf("lan%d dma xface error, ",
                                        unit);
                                printf("status = 0x%b, code = 0x%x\n",
                                        sifrbuf & 0xfff0,
                                        LAN_STAT_BITS,
                                        sifrbuf & 0x000f);
                        );
                        break;
                }
        }
}
```

```
                /* end handshake */
                if (success)
                        return (0);
        }
        /* retry init procedure */
        printf("lan%d: token ring adapter initialization failure, status = 0x%b, error code = 0x%x.\n",
                unit, sifrbuf & 0xfff0, LAN_STAT_BITS, sifrbuf & 0x000f);
        return (LAN_INIT_ERROR);
}


 /* end lan_initialize */


/*
 * lan_setaddr - set adapter's internet address
 */
lan_setaddr(ifp, sin)
        register struct ifnet *ifp;
        register struct sockaddr_in *sin;
{
        ifp->if_addr = *(struct sockaddr *)sin;
        ifp->if_net = in_netof(sin->sin_addr);
        ifp->if_host[0] = in_lnaof(sin->sin_addr);
        sin = (struct sockaddr_in *) & ifp->if_broadaddr;
        sin->sin_family = AF_INET;
        sin->sin_addr = if_makeaddr(ifp->if_net, INADDR_ANY);
        ifp->if_flags ]= IFF_BROADCAST;
}


/*
 * close - terminate communication on ring
 */
lan_close(unit)
        register int unit;
{
        register struct lan_softc *lns = &lan_softc[unit];
        register struct lan_device *addr = (struct lan_device *)lan_info[unit]->iod_addr;
        register struct lan_ctl *ctl = lns->lns_ctl;

        lns->lns_if.if_flags &= ~IFF_UP;
        lns->lns_adapter &= ~(LAN_ADAP_OPEN ] LAN_OPEN_IN_PROGRESS);
        lan_exec(LAN_CLOSE, ctl->close_parm, addr, unit);
}


/*
 * freeze - freeze adapter to enable internal storage dump
 */
lan_freeze(unit)
        register int unit;
{
        register struct lan_softc *lns = &lan_softc[unit];
        register struct lan_device *addr = (struct lan_device *)lan_info[unit]->iod_addr;
        register int s = splimp();
        register int i;
        lns->lns_adapter ]= LAN_ADAP_FROZEN;
        lns->lns_if.if_flags &= ~(IFF_RUNNING ] IFF_UP);
        /*
         * generate pulses on adapter reset line
         * to freeze; write to sif cmd reg for
         * microcode level 12 compatibility
         */
        for (i = 0; i < LAN_ADAP_FREEZE_PULSES; i++) {
                if (i)
                        DELAY(LAN_ADAP_FREEZE_DELAY);
                miow(&(addr->lan_hreset), 0);
                DELAY(LAN_ADAP_MIN_RESET);
                miow(&(addr->lan_hreset), 0);
        }
        *(char *)(&(addr->lan_cmdstat)) = LAN_FREEZE_SIG_u12;
        splx(s);
}


/*
 * unfreeze - unfreeze adapter
 */
lan_unfreeze(unit)
        register int unit;
{
        register struct lan_softc *lns = &lan_softc[unit];
        register struct lan_device *addr = (struct lan_device *)lan_info[unit]->iod_addr;
        register int s = splimp();

        /* pulses hardware reset line to unfreeze */
        miow(&(addr->lan_hreset), 0);
        DELAY(LAN_ADAP_MIN_RESET);
        miow(&(addr->lan_hreset), 0);
        lns->lns_adapter &= ~LAN_ADAP_FROZEN;
        splx(s);
}


/*
 * thaw - release adapter frozen state
 */
lan_thaw(unit)
        register int unit;
{
        register struct lan_softc *lns = &lan_softc[unit];

        if (lns->lns_adapter & LAN_ADAP_FROZEN) {
                lan_unfreeze(unit);
                timeout(lan_reset, (caddr_t)unit, 1 * hz);
        }
}


/*
 * lan_read_adapter - transfer adapter storage to system
 */
lan_read_adapter()
{
/* TO BE IMPLEMENTED */
}
```

```
/*
 * lan_error_log - read and reset adapter error log
 */
lan_error_log()
{
/* TO BE IMPLEMENTED */
}


/*
 * lan_set_faddr - reset adapter functional address after open
 */
lan_set_faddr()
{
/* TO BE IMPLEMENTED */
}

/*
 * lan_dma_setup - for a given virtual address, return a tcw entry
 *                 corresponding to the real address mapping
 */
lan_dma_setup(vaddr, type, chan)
        register char *vaddr;
        register int type;
        register int chan;

{

        register int raddr;
        register unsigned int i;
        register short *tcwp;

        raddr = vtop((int)vaddr & ~(PAGESIZE - 1));

        switch (type) {
        case TCW_RESERVE:
                /*
                 * if tcw is type RESERVE it will not be freed
                 * so try to allocate from end of table down
                 */
                tcwp = (short *)(TCW_BASE + (((chan << 6) + (NUMTCW - 1)) << 1));
                for (i = NUMTCW - 1; i >= 0; i--, tcwp--) {
                        if (*tcwp == TCW_FREE)
                                goto alloc_slot;
                        if ((((int) *tcwp &
                                        (PAGESIZE-1)) << 11) == raddr)
                                return (i);
                }
                return (TCW_ERROR);
        /*
         * if tcw is for single use we expect it to be freed
         * so allocate it from the start of the table up
         */
        case TCW_SINGLE_USE:
                tcwp = (short *)(TCW_BASE + (chan << 7));
                for (i = 0; i < NUMTCW; i++, tcwp++) {
                        if (*tcwp == TCW_FREE)
                                goto alloc_slot;
                }
                return (TCW_ERROR);
        default:
                return (TCW_ERROR);
        }
alloc_slot:
        *(short *)(TCW_BASE + (((chan << 6) + i) << 1)) =
                (short)(((int)raddr) >> 11) ] RSC_ACC ] REAL_ACC;
        return (i);
}

/*
 * lan_dma_init - initialize the tcw table so all entries are free
 */

lan_dma_init(chan)
        register int chan;
{
        register short *tcwp;
        register int i;
        tcwp = (short *)(TCW_BASE + (chan << 7));
        for (i = NUMTCW; i > 0; i--, tcwp++)
                *tcwp = TCW_FREE;
}


/*
 * lan_xid_test - support logical link control type 1 operation
 */

lan_xid_test(m, i, unit)
        register struct mbuf *m;
        register int i;
        register int unit;
{
        register struct lan_softc *lns = &lan_softc[unit];
        register struct ifnet *ifp = &lns->lns_if;
        char *c;
        int j;

        switch (lns->rh[i]->llc_ctl) {
        case LAN_LLC_XID_CMD0:
        case LAN_LLC_XID_CMD1:
                c = mtod(m, char *);
                for (j = 0; j < LAN_L_XID_RESP; j++) {
                        *c++ = lan_xid_resp[j];
                }
                m->m_len = LAN_L_XID_RESP;
        case LAN_LLC_TEST_CMD0:
        case LAN_LLC_TEST_CMD1:
                lan_output_llc(ifp, m, lns->rh[i]->from_addr,
                        lns->rh[i]->llc_ctl, lns->rh[i]->dsap);
                return (lns->rh[i]->llc_ctl);
                break;
        default:
                return(0);
                break;
        }
}
```

```
/*
 * miow - swap bytes before port output
 */

miow(ioport, datawd)
        register unsigned short *ioport;
        register unsigned short datawd;
{
        *ioport = ((datawd >> 8) ] (datawd << 8));
}


/*
 * mior - swap bytes after port input
 */
unsigned short
mior(ioport)                            /* swap bytes after port input */
        register unsigned short *ioport;
{
        register unsigned short datawd;

        datawd = *ioport;
        return ((unsigned short)(datawd >> 8) ] (datawd << 8));
}


#endif NLAN > 0


/*
 * 5799-CGZ (C) COPYRIGHT IBM CORPORATION 1986
 * LICENSED MATERIALS - PROPERTY OF IBM
 * REFER TO COPYRIGHT INSTRUCTIONS FORM NUMBER G120-2083
 */
/* $Header: if_lanreg.h,v 2.7 86/07/08 14:12:51 Exp $ */
/* $Source: /usr/sys/if_lanreg.h,v $ */

#if !defined(lint) && !defined(LOCORE) && defined(RCS_HDRS)
static char *rcsidif_lanreg = "$Header: if_lanreg.h,v 2.7 86/07/08 14:12:51 Exp $";
#endif

/* Adapter register addresses */
struct lan_device {
        unsigned short lan_data;        /* system interface data reg */
        unsigned short lan_datai;       /* system interface data incr */
        unsigned short lan_address;     /* system interface address */
        unsigned short lan_cmdstat;     /* command/status register */
        unsigned short lan_enable;      /* enable adapter interrupts */
        unsigned short lan_undef;       /* undefined adapter address */
        unsigned short lan_disable;     /* disable adapter interrupts */
        unsigned short lan_hreset;      /* hard reset adapter */
};
```

```
/* IRQ12 interrupt reset (shared level interrupt) */
#define LAN_IRQ12       ((char *)0xF00006F4)

/* Adapter command/status register offset for probe */
#define LAN_CMDREG      0x6

/* Adapter locations for initialization and adapter check */
/* These values are written to the lan_address register */
#define LAN_INIT_DATAA  0x0200
#define LAN_ACHECK_DATA 0x05E0

/* System to Adapter Interrupts */
/* These values are written to the lan_cmdstat register */
#define LAN_RESET       0xFF80  /* reset adapter */
#define LAN_SSBCLEAR    0xA000  /* notify that status block available */
#define LAN_EXECUTE     0x9080  /* initiate command in command block */
#define LAN_RECVCONT    0x8480  /* request recv operation to continue */
#define LAN_RECVALID    0x8280  /* signal recv list suspension cleared */
#define LAN_XMTVALID    0x8180  /* signal xmit list suspension cleared */

/* Adapter to System Response */
/* These values are read from the lan_cmdstat register */
#define LAN_INT 0x0080  /* valid interrupt */
#define LAN_ADAP_INT    0x000E  /* adapter -> system interrupt code */
#define LAN_ACHECK      0x0000  /* unrecoverable adapter error */
#define LAN_IMPLFRC     0x0002  /* IMPL force mac frame received */
#define LAN_RINGSTAT    0x0004  /* ring status update */
#define LAN_SCBCLEAR    0x0006  /* system command block clear */
#define LAN_CMDSTAT     0x0008  /* command status update */
#define LAN_RECVSTAT    0x000A  /* receive status update */
#define LAN_XMITSTAT    0x000C  /* transmit status update */
#define LAN_STAT_BITS   \
"\20\20INTADAP\17RESET\16SSBCLR\15EXECUTE\14SCBREQ\13RCVCONT\12RCVLD\11XMTVLD\
\10INTSYS\7INIT\6TEST\5ERR"

/* System Command Block */
struct lan_scb {
        unsigned short command;
        unsigned short h_addr;
        unsigned short l_addr;
        };

/* Adapter Commands */
/* These values are written to the lan_scb control block */
#define LAN_OPEN        0x0003  /* open adapter */
#define LAN_TRANSMIT    0x0004  /* transmit frame */
#define LAN_TRANSHLT    0x0005  /* interrupt transmit list chain */
#define LAN_RECEIVE     0x0006  /* receive frames */
#define LAN_CLOSE       0x0007  /* close adapter */
#define LAN_SETGADDR    0x0008  /* set group address */
#define LAN_SETFADDR    0x0009  /* set functional address */
#define LAN_RDERRORLOG  0x000A  /* read error log */
#define LAN_RDADAPTR    0x000B  /* read adapter storage */

/* System Status Block */
struct lan_ssb {
        unsigned short command;
        unsigned short status0;
        unsigned short status1;
        unsigned short status2;
        };

/* Adapter Status */
/*      These values are read from the lan_ssb control block */
#define LAN_SSB_RING    0x0001  /* ring status update */
#define LAN_SSB_REJECT  0x0002  /* command reject */
#define LAN_CMD_REJ_BITS        \
"\20\20ILLCMD\17ADDRERR\16ADAPOPN\15ADAPCLS\14SAMCMD"

/* Adapter Check Status */
/* These values are read from the adapter check field */

#define ADAP_CHK_SIZE   4
#define RECV_PARITY     0x80
#define XMIT_PARITY     0x40
#define XMIT_UNDERRUN   0x20
#define RECV_OVERRUN    0x10
#define LAN_ACHECK_BITS \
"\20\20DIOPAR\17DMARD\16DMAWT\15ILLOP\14LBPAR\13EMPAR\12SIFPAR\11PHPAR\
\10RCVPAR\7WTPAR\6UNDRN\5OVRUN\4INVINT\3INVERR\2INVXOP\1PGMCHK"

/* Ring Status */
/* These values are read from the lan_ssb on ring status update */
#define LAN_SIGNAL_LOSS 0x8000  /* signal loss */
#define LAN_HARD_ERROR  0x4000  /* xmit/recv beacon frames */
#define LAN_SOFT_ERROR  0x2000  /* xmit report error mac frame */
#define LAN_XMIT_BEACON 0x1000  /* xmit beacon frames */
#define LAN_WIRE_FAULT  0x0800  /* short circuit in data path */
#define LAN_AUTOER1     0x0400  /* auto-removal process */
#define LAN_AUTOER2     0x0200  /* reserved */
#define LAN_REMOVE_RECV 0x0100  /* remove received */
#define LAN_CTR_OVER    0x0080  /* counter overflow */
#define LAN_SINGLE      0x0040  /* single station */
#define LAN_RING_BITS   \
"\20\20SIGLOSS\17HRDERR\16SFTERR\15BEACON\14WRFLT\13AERR\11RMV\8CTROVFL\7SNG"

/* Adapter Initialization Parameters */
#define LAN_INIT_OPTIONS        0x8000  /* resv bit on, default options */
#define LAN_INIT_CMDSTAT        0x0C0C  /* int vector cmd, xmit stat */
#define LAN_INIT_RECVRING       0x0C0C  /* int vector recv,ring stat */
#define LAN_INIT_SCBCHK 0x0C0C  /* int vector scb,adapter check */
#define LAN_INIT_RBURST 0x0000  /* dma burst size for recv data */
#define LAN_INIT_XBURST 0x0000  /* dma burst size for xmit data */
#define LAN_INIT_ABORT  0x0101  /* no. of dma attempts if error */

/* Adapter Initialization Status */
#define LAN_INITIALIZE  0x0040  /* bring-up diagnostics complete */
#define LAN_TEST        0x0020  /* initialization test */
#define LAN_ERROR       0x0010  /* initialization error */

/* Adapter Open Parameters */
#define LAN_OPEN_OPTIONS        0x0000  /* open options */
#define LAN_OPEN_RLIST          0x0000  /* receive list size */
#define LAN_OPEN_XLIST          0x0000  /* transmit list size */
#define LAN_OPEN_BUFSIZE        0x00e8  /* buffer size = 224 bytes */
#define LAN_OPEN_RAMSTART       0x4006  /* RAM start address */
#define LAN_OPEN_RAMEND 0x7FFE  /* RAM end address */
#define LAN_OPEN_XMINMAX        0x040e  /* xmit buffer min/max counts */
```

```
/* Open Status */
#define LAN_OPEN_COMPLETE       0x8000  /* open complete */
#define LAN_OPEN_NODE_ERROR     0x4000  /* node address error */
#define LAN_OPEN_LIST_ERROR     0x2000  /* recv/xmit list size error */
#define LAN_OPEN_BUF_ERROR      0x1000  /* buffer size error */
#define LAN_OPEN_RAM_ERROR      0x0800  /* RAM address error */
#define LAN_OPEN_XMIT_ERROR     0x0400  /* xmit buffer count error */
#define LAN_OPEN_ERROR          0x0200  /* error detected during open */
#define LAN_OPEN_STAT_BITS      \
"\20\20OPENOK\17ADDRERR\16LSTSZ\15BUFSZ\14RAMERR\13XMTBFCT\12OPENERR"
char *open_errmsg[16] = {
        "undefined",
        "function failure",
        "receiver exception",
        "undefined",
        "undefined",
        "timeout",
        "ring failure",
        "ring beaconing",
        "duplicate node address",
        "request parameters",
        "remove received",
        "IMPL force received" };

/* Open Command Phases */
#define LAN_OPEN_LOBE_TEST      0x0010  /* lobe media test */
#define LAN_OPEN_INSERTION      0x0020  /* physical insertion */
#define LAN_OPEN_ADDR_VER       0x0030  /* address verification */
#define LAN_OPEN_ROLL_CALL      0x0040  /* roll call poll */
#define LAN_OPEN_REQ_PARM       0x0050  /* request parameters */

/* Open Error Codes */
#define LAN_OPEN_FUNC_FAILURE   0x0201  /* function failure */
#define LAN_OPEN_OSIGNAL_LOSS   0x0202  /* signal loss */
#define LAN_OPEN_OWIRE_FAULT    0x0203  /* wire fault */
#define LAN_OPEN_FREQ_ERROR     0x0204  /* unused */
#define LAN_OPEN_TIMEOUT        0x0205  /* timeout */
#define LAN_OPEN_RING_FAILURE   0x0206  /* ring failure */
#define LAN_OPEN_RING_BEACON    0x0207  /* ring beaconing */
#define LAN_OPEN_DUP_NODE       0x0208  /* duplicate node address */
#define LAN_OPEN_OREQ_PARM      0x0209  /* request parameters */
#define LAN_OPEN_OREM_RECV      0x020A  /* remove received */
#define LAN_OPEN_OIMPL          0x020B  /* IMPL force received */

/* Close Status */
#define LAN_CLOSE_COMPLETE      0x8000  /* close complete */

#define LAN_L_ADDR      6       /* length lan address */
#define LAN_N_DATA      3       /* max data fields in recv/xmit list */
/* Receive, Transmit Lists */
struct lan_list {
        unsigned short xlp_h;
        unsigned short xlp_l;
        unsigned short cstat;
        unsigned short frame_size;
        struct d_list {
                unsigned short d_cnt;
                unsigned short d_haddr;
                unsigned short d_laddr;
        } d_parm[LAN_N_DATA];
};

/* Lan header includes control fields, source and destination
 addresses, and llc fields: */
struct list_hdr {
        char pcf0;
        char pcf1;
        char to_addr[LAN_L_ADDR];
        char from_addr[LAN_L_ADDR];
        char dsap;
        char ssap;
        char llc_ctl;
};

/* Receive Status */
#define LAN_FRAME_COMPLETE      0x8000  /* received frame complete */
#define LAN_RECV_SUSPEND        0x4000  /* receive chain ended */
#define LAN_RCSTAT_COMPLETE     0x4000  /* received frame complete */
#define LAN_RECV_BITS           "\20\20RCVCMPL\17RCVSUSP"
#define LAN_RCSTAT_BITS "\20\17FRMCMPL\16FRMSTRT\15FRMEND"


/* Transmit Status */
#define LAN_XCSTAT_COMPLETE     0x4000  /* transmitted frame complete */

/* Adapter Storage for Burned-In Address */
struct lan_bia {
        unsigned short count;
        unsigned short adap_addr;
        char bia[6];
        char flag;
};

/* Miscellaneous */
#define LAN_RETRY       4       /* Retries during initialization */
#define  LAN_RESET_WAIT 300     /* Wait time (in 10's of ms) for bring-up-diags */
#define  LAN_DMA_TIMEOUT        1000    /* Dma timeout error (10's of ms)       */
#define  SCB_LEN        6       /* Number bytes in scb                  */
#define  SCB_INIT       0x0000c1e2d48b  /* Scb initialization contents  */
#define  SSB_LEN        8       /* Number bytes in ssb                  */
#define  SSB_INIT       0xffffd1d7c5d9c3d4      /* Ssb initialization contents */
#define LAN_ADDRESSES   0x0a04  /* Pointer to adap addresses in adap storage */
#define LAN_OPEN_PHASE_MASK     0xff0f  /* Ignore phase if open error */
#define LAN_NUM_IPARMS  11      /* Number of initialization parameters */
#define LAN_TIMEOUT     10      /* Dma timeout in 10 seconds */
#define LAN_PCF0        0x00    /* Physical Control Field 0 */
#define LAN_PCF1        0x40    /* Physical Control Field 1 (not mac) */
#define LAN_PCF1        0x40    /* Physical Control Field 1 (not mac) */
#define LAN_CHAIN       0x8000  /* chain indicator in recv/xmit list */
#define LAN_ODD_PTR     0x0001  /* end of list indicator */
#define LAN_X_CSTAT_REQ 0xB7FF  /* Xmit cmd/stat on request */
#define LAN_XMIT_VALID          0x8000  /* Xmit list valid indicator */
#define LAN_XMIT_EOF            0x1000  /* Xmit end of frame indicator */
#define LAN_RECV_ALT    1       /* Multiple received frames reported on intr */
#define LAN_R_CSTAT_REQ 0x88FF  /* Recv cmd/stat on request */
```

```
/*      Settings to en/disable the dma channels used by the adapter     */
#define  CH5_PAGE       0x04
#define  CH6_PAGE       0x02
#define  CH_ENABLE      0x00
#define  CH_DISABLE     0x04
#define  CH5            0x01
#define  CH6            0x02

/*      TCW Variables                                                   */
#define  NUMTCW 64
#define  TCW_FREE       0xffffffff      /* in raddr means tcw slot free */
#define  TCW_SINGLE_USE 1               /* tcw freed after single dma op*/
#define  TCW_RESERVE            2       /* tcw will remain in use*/
#define  TCW_ERROR              -1      /* error in tcw allocation*/
#define  ALTMAST_PAGEMODE_HIBITS        0xfe
#define  ALTMAST_PAGEMODE_DISP          0x7ff
#define  ALTMAST_PAGEMODE_TCW_LOBITS    11
#define  ALTMAST_PAGEMODE_TCW_HIBIT     5

/*      Address conversion macros                                       */
#define  PAGESIZE       2048
#define  LOG2PAGESIZE   11
#define  DMA_HI_ADDR(x) (ALTMAST_PAGEMODE_HIBITS ] (x >> ((sizeof(u_short) << 3)-LOG2PAGESIZE)))
#define  DMA_LO_ADDR(y,x) (((int)y & (PAGESIZE - 1)) ] (x << LOG2PAGESIZE))


/*
 * 5799-CGZ (C) COPYRIGHT IBM CORPORATION 1986
 * LICENSED MATERIALS - PROPERTY OF IBM
 * REFER TO COPYRIGHT INSTRUCTIONS FORM NUMBER G120-2083
 */
/* $Header: if_lanvar.h,v 2.9 86/07/08 14:12:58 Exp $ */
/* $Source: /usr/sys/if_lanvar.h,v $ */

#if !defined(lint) && !defined(LOCORE)  && defined(RCS_HDRS)
static char *rcsidif_lanvar = "$Header: if_lanvar.h,v 2.9 86/07/08 14:12:58 Exp $";
#endif

struct lan_ctl {
        struct lan_scb lan_scb;
        struct lan_ssb lan_ssb;
        struct lan_bia lan_bia;
        unsigned short open_parm[16];
        unsigned short close_parm[2];
};

#define LAN_XMITLIST_CT 2       /* Number of transmit lists */
#define LAN_RECVLIST_CT 2       /* Number of receive lists */
#define LAN_MAX_LISTS_PER_PAGE  75      /* Number of lists per big mbuf */

struct lan_recv_ctl{
        struct lan_list lan_rlist[LAN_RECVLIST_CT];
        struct list_hdr list_hdr[LAN_RECVLIST_CT];
};

/* Token-Ring software status per adapter */
struct lan_softc {
        struct arpcom lns_ac;           /* like ethernet structs */
#define lns_if lns_ac.ac_if             /* network-visible interface */
#define lns_addr lns_ac.ac_enaddr       /* hardware lan address */
        short lns_oactive;              /* is output active */
        short lns_xbuf;                 /* in-use xmt buf */
        short lns_nextbuf;              /* next buf to fill */
        short lns_xstart[LAN_XMITLIST_CT];
        short lns_ring;                 /* ring state */
        short lns_adapter;              /* adapter state */
        short lns_open_retries;         /* open retry count */
        short lns_beacon;               /* open retries while beaconing */
        short lns_ring_watch;           /* control timeout count */
        unsigned short lns_node_addr;   /* node address in adapter storage */
        struct proc *lns_freezer;
        struct lan_ctl *lns_ctl;        /* control structure pointer */
        struct lan_list *xp[LAN_XMITLIST_CT];/* xmit list ptrs */
        struct lan_list *rp[LAN_RECVLIST_CT];/* recv list ptrs */
        struct list_hdr *rh[LAN_RECVLIST_CT];/* recv hdr ptrs */
        struct lan_recv_ctl *lns_recv;  /* recv struct pointer */
        char *lns_xbufp[LAN_XMITLIST_CT];
        struct mbuf *lns_rbufp[LAN_RECVLIST_CT];        /* receive buffer pointers */
        struct mbuf *lns_rdata[LAN_RECVLIST_CT];        /* receive data area pointers */
        struct lan_tcw_list {           /* remember tcw's in use */
                int num_entries;
                int tcw_slot[NUMTCW];
                } lan_tcw_list[LAN_XMITLIST_CT];
        int lan_dma_chan;               /* dma channel used by this addr*/
};

/* Adapter/Ring Status */
#define LAN_ADAP_OPEN           0x8000  /* Adapter open */
#define LAN_OPEN_IN_PROGRESS    0x4000  /* Adapter open in progress */
#define LAN_RETRY_IN_PROGRESS   0x2000  /* Adapter open being retried */
#define LAN_ADAP_BROKEN         0x1000  /* Adapter failure */
#define LAN_ADAP_AUTOER1        0x0800  /* Adapter internal error */
#define LAN_ADAP_FCTNFAIL       0x0400  /* Adapter function failure */
#define LAN_ADAP_BIA_READ       0x0200  /* Adapter node address read */
#define LAN_ADAP_DOWN           0x0100  /* Adapter closed */
#define LAN_ADAP_FROZEN         0x0008  /* Adapter frozen for dump */
#define LAN_BEACONING           0x0001  /* Ring beaconing */
#define LAN_RECOVERY            0x0002  /* Ring in recovery */
#define LAN_CABLE_FAIL          0x0004  /* Cable failure */

/* Logical Link Control Class 1 Definitions */
#define LAN_IPTYPE      0x06    /* IP packet type */
/* TEMP:  FOLLOWING FAKE VALUE IS PENDING ARP NUMBER ASSIGNMENT */
#define LAN_ARPTYPE     0x99
/* NB: The bits in the following Unnumbered Information Command are in the
 *      wrong order.  However, this sequence is being temporarily maintained
 *      for compatibility;  it should be corrected when all workstations are
 *      able to receive frames correctly when the UI command is sent correctly. */
#define LAN_UI_CMD      0xc0
#define LAN_LLC_XID_CMD0        0xaf
#define LAN_LLC_XID_CMD1        0xbf
#define LAN_LLC_TEST_CMD0       0xe3
#define LAN_LLC_TEST_CMD1       0xf3
#define LAN_L_XID_RESP          3
```

```
/* Maximum transmission unit */
#define LAN_MTU ETHERMTU        /* same as ethernet */

/* Miscellaneous */
#define  LAN_MAX_OPEN_RETRY     3       /* Max number of open retries   */
#define  LAN_INIT_ERROR -1
#define ONESEC          240000  /* argument to DELAY 1 second */
#define TEN_MS          2400    /* argument to DELAY 10 ms */
#define SIXTY           60      /* ring recovery time in seconds */
#define LAN_ADDR_PENDING        0x01    /* Waiting to update lan addr */


/*
 * 5799-CGZ (C) COPYRIGHT IBM CORPORATION 1986
 * LICENSED MATERIALS - PROPERTY OF IBM
 * REFER TO COPYRIGHT INSTRUCTIONS FORM NUMBER G120-2083
 */
/* $Header: if_lanio.h,v 1.1 86/05/20 11:11:37 Exp $ */
/* $Source: /usr/sys/if_lanio.h,v $ */

#if !defined(lint) && !defined(LOCORE)  && defined(RCS_HDRS)
static char *rcsidif_lanio = "$Header: if_lanio.h,v 1.1 86/05/20 11:11:37 Exp $";
#endif

/*      Adapter Storage for Freeze-Dump Data                            */
#define  LAN_FREEZE_DUMP 4096   /* Size of freeze-dump data in bytes    */
#define  LAN_FREEZE_CHUNK 2048  /* Size of read window when frozen      */
#define  LAN_FREEZE_INCR        0xff00  /* Increment addr during freeze */
#define  LAN_ADAP_FREEZE_PULSES 2       /* Number pulses to freeze      */
#define  LAN_ADAP_FREEZE_DELAY  120     /* Delay between pulses to detect*/
#define  LAN_ADAP_MIN_RESET     4       /* Delay between back-to-back dio*/
#define  LAN_FREEZE_SIG_u12     0x80    /* ucode level 12 compatibility */
struct lan_dump {
        int lan_len;            /* buffer length        */
        unsigned short lan_dump_data[LAN_FREEZE_DUMP/2];
        };

/* lan freeze-dump controls */
#define SIOCFLANDUMP    _IOW(i, 128, struct ifreq)      /* freeze lan */
#define SIOCSLANDUMP    _IOW(i, 127, struct ifreq)      /* set lan dump */
#define SIOCGLANDUMP    _IOW(i,126, struct ifreq)       /* get lan dump */
```

Index

RT PC Technical Reference
Token-Ring Network Adapter

Publication No. SK2T-0291-1

READER'S COMMENT FORM

This manual is part of a library that serves as a reference source for systems analysts, programmers, and operators of IBM systems. You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

Note: Copies of IBM publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.

Possible topics for comment are:

Clarity Accuracy Completeness Organization Coding Retrieval Legibility

If you wish a reply, give your name, company, mailing address, and date:

____

____

____

____

What is your occupation? ______________________________

Number of latest Newsletter associated with this publication: ____________________

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments or you may mail directly to the address in the Edition Notice on the back of the title page.)

Fold and tape

Please Do Not Staple

Fold and tape

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 40 ARMONK, N.Y.

POSTAGE WILL BE PAID BY ADDRESSEE

International Business Machines Corporation
Dept. E02
P.O. Box 12195
Research Triangle Park, N.C. 27709-9990

International Business
Machines Corporation
Department E02
P.O. Box 12195
Research Triangle Park, NC 27709

Printed in the
United States of America

SK2T—0291—1