

**EXOS/101
ETHERNET FRONT-END PROCESSOR
REFERENCE MANUAL**

**Excelan, Inc.
2180 Fortune Drive
San Jose, CA 95131**

August 1, 1983



NOTICE

This document reflects the features and specifications of the EXOS/101, and the NX/101 firmware version 2.n. Excelan, Inc. reserves the right to make changes and improvements in features and specifications at any time without prior notice or obligation.

The following are trademarks or equipment designations of Excelan, Inc.:

EXOS
EXOS/101
NX
NX/101

Ethernet is a trademark of Xerox Corporation.

Multibus is a trademark of Intel Corporation.

Copyright © 1982 by Excelan Inc. All rights reserved. No part of this document may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior permission of Excelan.



PREFACE

This document describes the EXOS/101 Ethernet Front-End Processor board. It covers information necessary to integrate the EXOS/101 in a Multibus-based system, and to design software both for the host and the EXOS/101. Ethernet and Multibus are described in readily available documents; this manual makes no special effort to explain these standards.

By design, the EXOS/101's operating system kernel (NX/101) insulates user protocol software from hardware implementation details. This approach simplifies software design, and facilitates portability to future products which will take advantage of VLSI Ethernet controllers. Therefore this manual primarily describes the NX/101 kernel, with reference to hardware design only where necessary. It is intended only as a reference manual, and does not undertake to explain the product's design philosophy. Separate manuals, available from Excelan, discuss the concepts which guided the design of the EXOS/101.

Section 1 of this manual outlines the principle features of the EXOS/101.

Section 2 is a guide to useful references.

Section 3 describes conventions and restrictions which are crucial to successful application of the EXOS/101.

Section 4 describes initialization of the EXOS/101, including software download from the host.

Section 5 discusses using the EXOS/101 as an intelligent link level controller. In this mode, no software is down-loaded, so that only glancing reference to sections 6 through 9 will be necessary.

Sections 6, 7, and 8 describe the NX/101 firmware, which provides support for software down-loaded to the EXOS/101. Section 6 describes the real-time, multitasking OS kernel services, and describes the programming environment aboard the EXOS/101. Sections 7 and 8 cover the Ethernet and host interface facilities, which are implemented in NX/101. They are broken out into separate chapters because NX/101's design makes them conceptually detachable.

Section 9 defines the NX/101 kernel calls, and is intended for ready reference once NX/101 services are understood functionally.

Section 10 describes the EXOS/101's network bootstrap protocol, which can be used to automatically down-load software to the EXOS/101 over the Ethernet at initialization time.

Section 11 provides necessary information about EXOS/101 hardware.

TABLE OF CONTENTS

1.	INTRODUCTION	1
1.1.	Overview	1
1.2.	EXOS/101 Hardware Description	2
1.3.	NX/101 Firmware Description	6
2.	REFERENCES	10
3.	NOTATIONS AND CONVENTIONS	11
3.1.	Number Base	11
3.2.	Data Object Terminology	11
3.3.	Message Format Specification	11
3.4.	Procedural Specifications	11
3.5.	Bit Position and Value Specifications	12
3.6.	Data Storage Order	12
3.7.	Integration with 68000-Based Systems	13
3.8.	Data Alignment	13
3.9.	Memory Address Format	13
3.10.	Shared Multibus Memory Access Restrictions	15
4.	INITIALIZATION AND HOST INTERFACE	16
4.1.	Hardware Communications Facilities	17
4.2.	Host Data Order Conversion Option	18
4.3.	Reset and Configuration Procedure	20
4.4.	Configuration Message Format	23
4.5.	Message Queue Format	31
4.6.	Message Queue Initialization	33
4.7.	Message Queue Protocol	34
4.8.	Down-Loading Software from the Host	38
5.	LINK LEVEL CONTROLLER MODE	42
5.1.	The Controller Mode Interface	42
5.2.	TRANSMIT Request/Reply Message	46
5.3.	RECEIVE Request/Reply Message	49
5.4.	NET_MODE Request/Reply Message	52
5.5.	NET_ADDRS Request/Reply Message	55
5.6.	NET_RECV Request/Reply Message	58
5.7.	NET_STSTCS Request/Reply Message	60
6.	THE NX/101 PROGRAMMING ENVIRONMENT	63
6.1.	Overview	63
6.2.	Memory Organization	63
6.3.	Interrupt Types	65
6.4.	Processes	66
6.5.	Mailboxes	68

6.6.	Process Locks	70
6.7.	System Mailboxes	70
6.8.	The Clock Device	72
7.	THE NX/101 ETHERNET INTERFACE	73
7.1.	Ethernet Transmit Request	73
7.2.	Ethernet Receive Request	76
7.3.	Ethernet Controller Modes	79
7.4.	Ethernet Controller Option Mask	80
7.5.	Address Slots	80
7.6.	Net Statistics	81
8.	THE NX/101 HOST INTERFACE	83
8.1.	Host Transmit Request	83
8.2.	Host Receive Request	85
8.3.	Direct Access to Host System Memory	86
8.4.	Host Data Order Conversion	86
9.	NX/101 KERNEL CALL REFERENCE	88
10.	INITIALIZING AND DOWN-LOADING FROM THE ETHERNET	116
10.1.	Network Bootstrap Protocol Description	116
10.2.	Data Transmission Order	123
10.3.	Network Bootstrap Protocol Message Header	123
10.4.	Message Encapsulation	125
10.5.	FIND and SELECT Request/Reply Messages	126
10.6.	DOWNLOAD Request/Reply Message	129
10.7.	UPLOAD Request/Reply Message	131
10.8.	CONFIGURE Request/Reply Message	132
10.9.	EXECUTE Request/Reply Message	133
11.	HARDWARE REFERENCE	135
11.1.	Access to EXOS/101 Components	135
11.2.	Multibus Interface	135
11.3.	Ethernet Interface	138
11.4.	On-Board Processing Capabilities	140
11.5.	Firmware Configuration Options	141
11.6.	Self-Test Operation	141
11.7.	Power Requirements	141
11.8.	Operating Environment	142

APPENDICES

APPENDIX A: EXOS/101 Component Location

FIGURES

Figure 1-1:	An EXOS/101 Front-End Processor Mode Implementation	2
Figure 1-2:	EXOS/101 Block Diagram	3
Figure 1-3:	NX/101 Software Architecture	6
Figure 3-1:	Mapping of Segmented Address into Longword Data Type	14
Figure 3-2:	Mapping of Absolute Address into Longword Data Type	14
Figure 4-1:	Host Data Order Conversion Option Test Pattern	19
Figure 4-2:	Host Data Format Test Pattern Initialization	20
Figure 4-3:	Typical Reset and Configuration Procedure	22
Figure 4-4:	Configuration Request/Reply Message	24
Figure 4-5:	Message Buffer Format	32
Figure 4-6:	Message Queue Data Structures at Initialization Time	34
Figure 4-7:	Example EXOS-to-Host Message Queue, at Initialization	35
Figure 4-8:	EXOS/101 Down-Load Request/Reply Message	39
Figure 4-9:	EXOS/101 Start-Execution Request/Reply Message	40
Figure 5-1:	Encapsulation of Request/Reply Message in Message Buffer	43
Figure 5-2:	Link Level Controller Mode Request Processing Scheme	44
Figure 5-3:	TRANSMIT Request/Reply Message	46
Figure 5-4:	RECEIVE Request/Reply Message	49
Figure 5-5:	NET_MODE Request/Reply Message	52
Figure 5-6:	NET_ADDRS Request/Reply Message	55
Figure 5-7:	NET_RECV Request/Reply Message	58
Figure 5-8:	NET_STSTCS Request/Reply Message	60
Figure 6-1:	Default EXOS/101 Memory Allocation	64
Figure 6-2:	Standard Header for System Messages	71
Figure 7-1:	Ethernet Packet Format	74
Figure 7-2:	Ethernet Transmit Request/Reply Message	75
Figure 7-3:	Ethernet Receive Request/Reply Message	78
Figure 8-1:	Host Transmit Request/Reply Message	84
Figure 8-2:	Host Receive Request/Reply Message	85
Figure 10-1:	State Diagram of Network Bootstrap Protocol	118
Figure 10-2:	Network Bootstrap Protocol Request/Reply Message Header	124
Figure 10-3:	Encapsulation of Request/Reply Message	126
Figure 10-4:	Network Bootstrap FIND/SELECT Request/Reply Message	127
Figure 10-5:	Network Bootstrap DOWNLOAD Request/Reply Message	130
Figure 10-6:	Network Bootstrap UPLOAD Request/Reply Message	131
Figure 10-7:	Network Bootstrap CONFIGURE Request/Reply Message	133
Figure 10-8:	Network Bootstrap EXECUTE Request/Reply Message	134
Figure 11-1:	Quick Reference to Jumper Options	136
Figure 11-2:	Port Address, Interrupt Level, NX/101 Option Jumpers	137
Figure 11-3:	Quick Reference to Status LEDs	138
Figure 11-4:	Self-Diagnostic and Configuration Error Codes	142



1. INTRODUCTION

This section provides an overview of the EXOS/101's features and specifications, and describes its principal modes of operation.

1.1. Overview

The EXOS/101 is a high-performance, front-end communications processor that connects a Multibus system to an Ethernet local area network. It implements the complete Ethernet Data Link Level interface, with significant functional extensions, on a single Multibus board. In addition, the EXOS/101 can support high-level network protocols on-board, thereby offloading this burden from the host CPU.

The EXOS/101 is directly managed by an on-board 8088 CPU, which runs the NX/101 operating system, stored in a 16 Kbyte EPROM. A host system controls the EXOS/101 primarily through command and reply messages located in memory accessible from the Multibus. NX/101 firmware interprets the command messages and generates the replies.

NX/101 provides two basic modes of operation, selected at initialization time. Link level controller mode is useful for applications where host-resident protocol software has already been developed, or where it is otherwise not feasible to download high-level protocols to run on the EXOS/101. Instead, NX/101 firmware brings the EXOS/101's Data Link controller functions out to the host interface. The host system obtains Data Link services through standard request/reply messages; the EXOS/101's RAM is entirely available for buffering packets.

In front-end processor mode, the host system downloads protocol software to the EXOS/101 at initialization time (or the EXOS/101 bootstraps itself from the Ethernet). This software then uses NX/101's real-time, multi-tasking process management services and I/O drivers to control the EXOS/101's Ethernet interface and manage communications with the host system. Standard protocol modules for the EXOS/101, such as the DARPA TCP/IP protocols, are available from Excelan. Figure 1-1 shows such an implementation in relation to the ISO Open Systems Integration model.

Alternately, users can develop, or port, their own protocols to run on the EXOS/101 under NX/101. This manual contains all information required to write software for the EXOS/101. NX/101 is designed to greatly facilitate this process.

First, NX/101 provides a set of parameterized mechanisms that reduce the development effort required for implementation of high level protocol software. This is accomplished by offering a multitasking environment and integrated drivers that provide high level primitives for the functions associated with the Ethernet controller, the host link, and the clock.

Another objective of NX/101 is to hide the implementation details of EXOS/101 hardware from user software by providing suitable abstractions for all facilities. Thus, user software written to NX/101 specifications will be directly compatible with future Excelan products that exploit LSI technology for the network controllers.

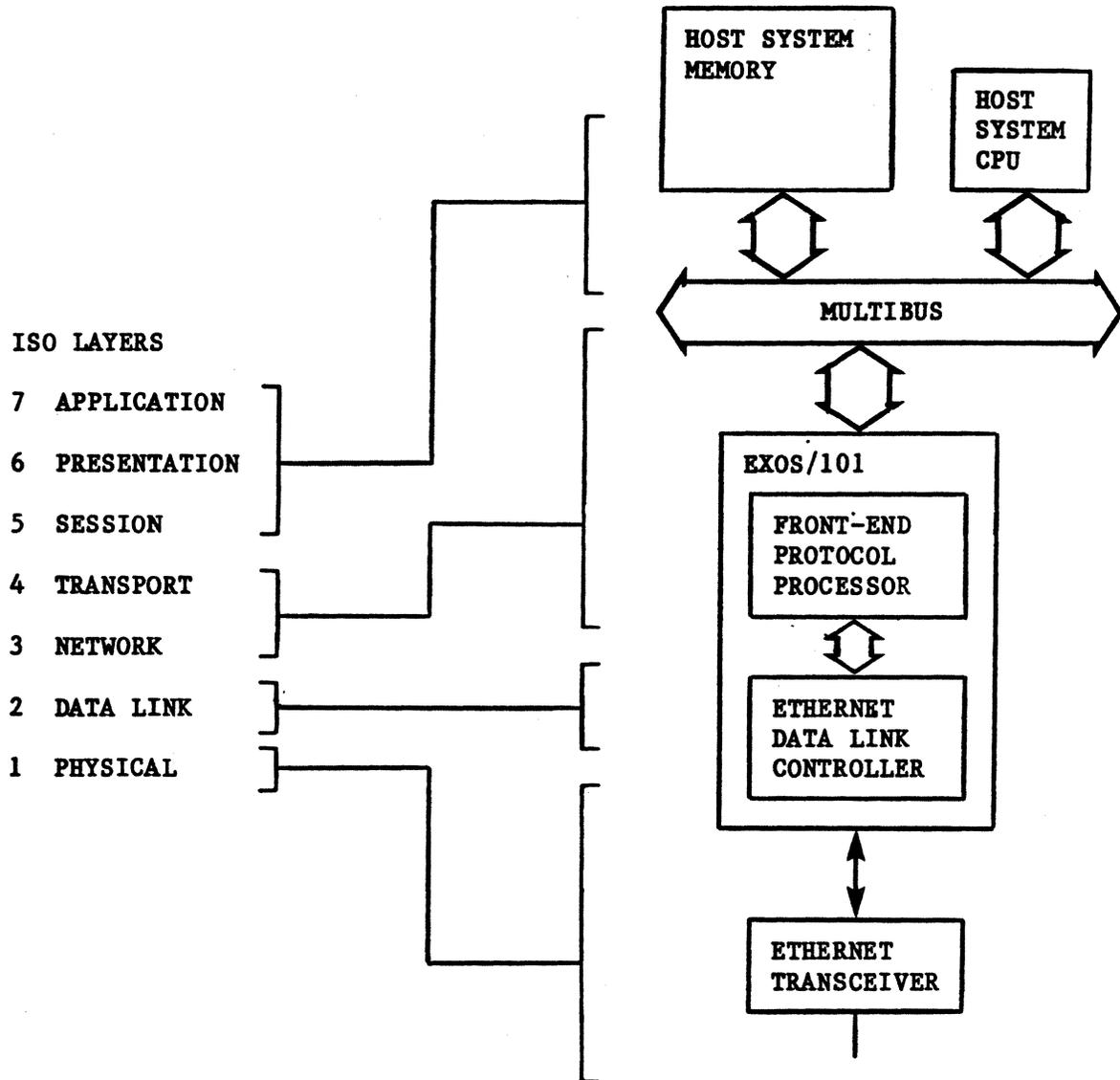


Figure 1-1: An EXOS/101 Front-End Processor Mode Implementation

1.2. EXOS/101 Hardware Description

Figure 1-2 shows a block diagram of the EXOS/101. Architecturally, the EXOS/101 consists of two loosely-coupled elements: an Ethernet Data Link Level controller, and a microprocessor-based protocol processing engine. These components communicate with each other through an internal bus and 64 Kbytes of dual-ported RAM. Components of the protocol processing engine occupy the left half of the block diagram; the Ethernet controller occupies the right half.

The EXOS/101 implements the Ethernet Data Link protocol almost entirely in a bipolar finite state machine. Functions such as address recognition, CRC

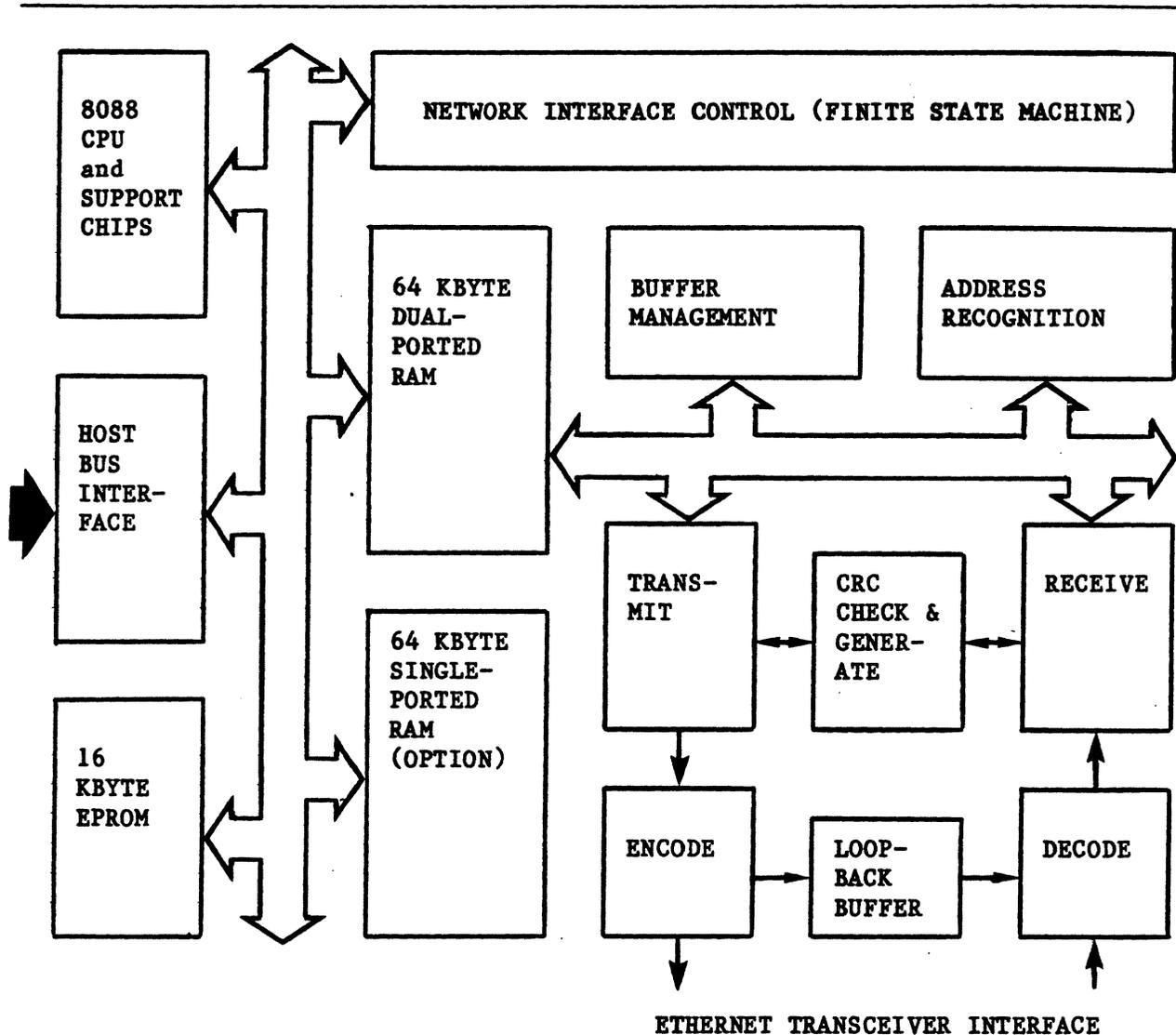


Figure 1-2: EXOS/101 Block Diagram

check, and buffer chaining are managed in hardware, so that the 8088 CPU is fully available for front-end processing applications. The protocol processing engine is supported by either 64K (Model 1) or 128K (Model 2) of RAM, a counter/timer circuit, and an interrupt controller. A 16 Kbyte EPROM contains Excelan's NX/101 firmware, which includes self-diagnostic tests, an operating system kernel, and network bootstrap code.

1.2.1. Principal Features

- o 12 by 6.75-inch card requires just one Multibus slot.
- o On-board 8 MHz 8088 microprocessor (6.67 MHz clock) and 64 Kbytes of RAM (128 Kbytes on Model 2) support high-level network protocols on-board.

- o Dual-ported memory allows concurrent, full-speed access by network hardware and on-board processor.
- o Can receive successive frames with minimum interframe spacing (9.6 microsec.). Can receive immediately after transmitting, or vice versa, with minimum interframe spacing and without losing data.
- o Hardware recognition of physical, broadcast, and 252 multicast addresses, in addition to promiscuous mode.
- o Hardware-supported buffer chaining allows buffering of up to 32 received frames without any CPU intervention. Allocation of buffers, both location and size, is completely under software control.
- o Time domain reflectometry (TDR) function helps diagnose cable faults. Resolution is 100 nsec.

1.2.2. Ethernet Compatibility

The EXOS/101 conforms fully with Ethernet version 1.0 specification published by DEC, Intel and Xerox on September 30, 1980. Integrated with a standard Ethernet transceiver, it provides all Data Link and physical layer services.

1.2.3. Multibus Compatibility

The EXOS/101 conforms with IEEE 796 (Multibus) specifications, as an 8-bit master. Compliance is D8 M24 I16 V0 L (8-bit transfers, 24-bit addressing, and non-bus vectored interrupts).

1.2.4. Multibus Interface

The EXOS/101 can access the entire Multibus system memory space (16 Mbyte) and the full 64K I/O space, as an 8-bit bus master. An additional one-byte communication path is provided from the Multibus to the EXOS processor via an I/O port. This can be used during initialization to transmit the address of a communication area in the shared Multibus memory.

The EXOS/101 and host processors can interrupt each other. The board generates non-bus vectored interrupts to interrupt the host. Interrupt priority can be set from INT0 to INT7, via jumper selection. The EXOS/101 provides a status bit, in case interrupt polling is required. The host can interrupt the EXOS/101 processor by writing to an I/O port.

1.2.5. Ethernet Functions

The EXOS/101 performs all physical and Link Level Ethernet functions except for transceiver functions. These include:

- o serial/parallel and parallel/serial conversion.
- o address recognition.

- o framing and unframing of messages.
- o Manchester encoding and decoding.
- o preamble generation and removal.
- o carrier sense and deference.
- o collision detection and enforcement, including jamming, backoff timing and retry.
- o FCS (CRC) generation and verification.
- o error detection and handling.

1.2.6. Address Recognition

Each board has a unique 48-bit Ethernet address, which is stored in EPROM (host software can override this address at run time). Recognition of physical, broadcast and multicast addresses is fully supported. Up to 252 multicast addresses can be assigned to a station; a very efficient filtering scheme reduces processing overhead. The EXOS/101 also provides a promiscuous mode, in which it accepts all addresses.

1.2.7. Frame Format

Link level frames are formatted as per the Ethernet specification, with 64 bits of synchronizing sequence (preamble), destination address (48 bits), source address (48 bits), message type (16 bits), data (46 to 1500 bytes) and FCS (32 bits). The preamble is generated and removed in hardware. Generation and checking of the Frame Check Sequence (FCS) is also handled in hardware.

1.2.8. Error Handling

The EXOS/101 handles all Ethernet error conditions, including CRC, alignment, and length errors. Packets containing these errors can optionally be received.

1.2.9. High Level Protocol Support

On-board processing power supports execution of higher level communications protocols, beyond the Ethernet link level. The elements of this programming environment are:

- o 8 Mhz 8088 CPU, operating at 6.67 MHz.
- o 64K dual-ported RAM (plus 64K single-ported RAM on Model 2).
- o 16 Kbytes of EPROM, containing NX/101 firmware.
- o clock timer circuit and interrupt controller.

Firmware supplied with the board (the NX/101 Network Executive) provides simplified Ethernet and host interface device drivers, and a multi-tasking

environment for high-level network protocols.

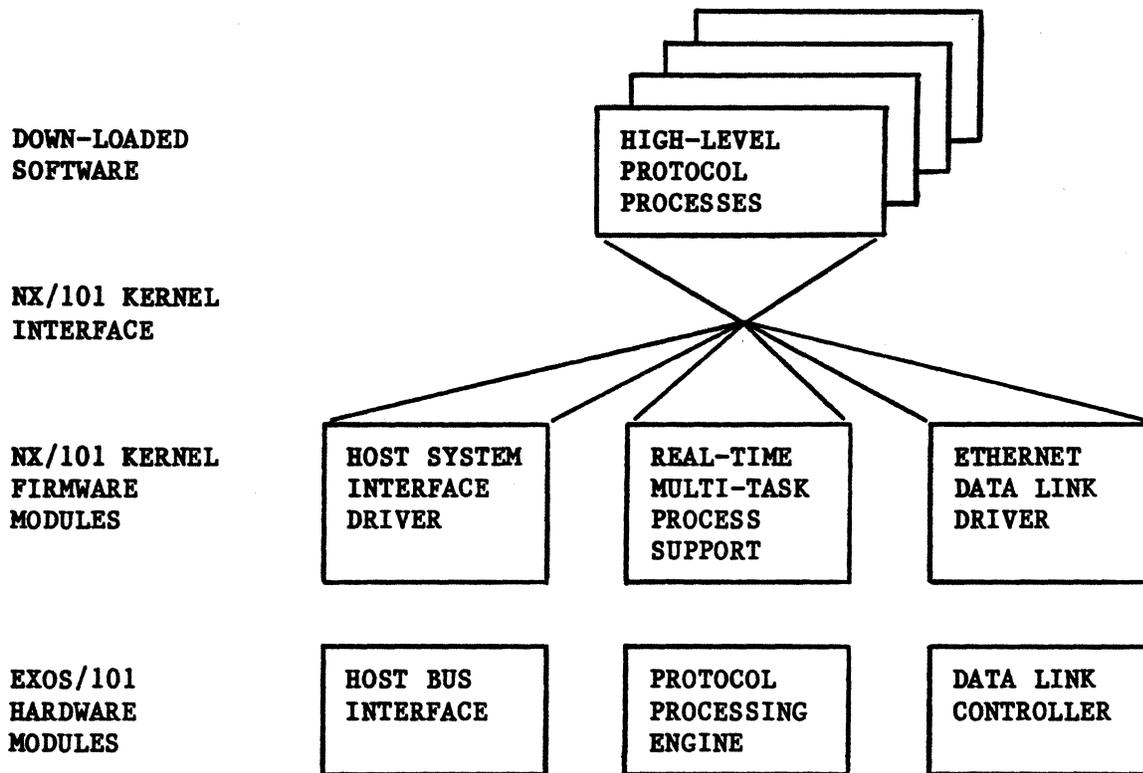


Figure 1-3: NX/101 Software Architecture

1.3. NX/101 Firmware Description

NX/101 resides in EPROM memory, which appears at the high end of the 1M byte address space of the 8088. NX/101 data structures use 4K of the RAM space; the rest is available for higher level software.

1.3.1. Principle Features

- o Self-diagnostics for testing the integrity of EXOS/101 hardware.
- o Booting process that allows higher level software to be down-loaded either from the host or from the network.
- o A real-time kernel that provides a multi-tasking environment, enabling the protocol software to be constructed in a structured manner as a set of cooperating processes.
- o Device drivers for the Ethernet controller and host computer interface. Access through message queues simplifies pipelined communications.

- o Supports network management functions by collecting network statistics.
- o Allows the EXOS/101 to be used as a simple Data Link controller, giving direct access to the network without down-loading any software.

1.3.2. Initialization

On reset the NX/101 firmware performs a series of self tests which confirm hardware integrity. In case of failure, the firmware communicates diagnostic codes through an LED display. After successful completion of the tests, the EXOS/101 either boots itself from the Ethernet, or awaits initialization by the host system, depending on a jumper option on the board.

If the jumper selects initialization by a host system, the host then uses a configuration message to select NX/101's mode of operation, and specify several other parameters. It can down-load software directly, tell NX/101 to boot itself from the Ethernet, or select link level controller mode. If initialization includes down-loading software, then NX/101 spawns a process and enters the front-end processor mode of operation. The following sections describe the execution environment for software which is down-loaded to the EXOS/101.

1.3.3. Multi-tasking support

NX/101 includes a real-time kernel that implements a multi-tasking environment for construction of higher level software in a structured manner. This kernel is fast by design, and imposes very little overhead. It supports two types of object - processes and mailboxes. The number supported of either object is configurable at start-up time.

A process is a unit of execution in the conventional sense. All processes share the same memory address space and can thus communicate via shared memory. Other than for NX/101's reserved memory there are no restrictions on how memory is used. Processes access NX/101 functions by executing the 8088's INT n instruction, where n identifies the service being requested.

A priority-based preemptive round robin scheduling algorithm allocates CPU time among processes. As many as 256 priority levels are supported, and the highest priority runnable process will always be scheduled. Among processes of the same priority, CPU time is allocated in time slices. A time slice is either infinity, or between 1 and 254 ticks, where each tick is 20 milliseconds. Any process can examine and change the priority and time slice of any process. Whether a process is runnable is determined solely by a sleep count, from 0 to 64K, and driven by the same clock as the time slice. Through this parameter, any process can suspend, delay or resume any process.

Interprocess communication and synchronization are implemented with mailboxes. Messages sent or received from the mailboxes can be either null or pointers to buffers in the common memory. Message buffer format is arbitrary except for the first field, which NX/101 uses to chain the messages in the mailbox queue. Sending and receiving of messages is fully synchronized. A process executing a receive call on a mailbox can specify the maximum time interval it is willing to wait. Waiting is implemented with the sleep count mechanism described above. If the specified time expires before a message arrives the process is

resumed and given an error code instead of a message. If only null messages are used, then the mailbox is identical to a conventional semaphore. The receive operation in this case is equivalent to the P operation and the send operation is equivalent to the V operation. The mailbox can be thus used as a synchronization mechanism both for a producer-consumer relationship and a critical section.

In addition to the mailbox, the NX/101 has a simpler and more efficient synchronization mechanism intended for short critical sections: the process lock. This operation postpones scheduling decisions until a corresponding unlock is executed, thereby excluding all other processes from running. Calls to lock can be nested up to 32K levels deep.

1.3.4. The Clock

NX/101's clock driver provides the abstraction of a 64-bit clock with a resolution of 20 milliseconds. Processes can read or set the time at will. On initialization the clock is set to zero.

1.3.5. Host Interface

NX/101 provides a uniform interface to the host regardless of the nature of the actual hardware host interface. The abstraction of the host is presented as a mailbox and read/write operations on host memory. The mailbox acts as a source and sink for messages and also provides synchronization between the processes on the host and the processes on the EXOS/101.

This interface appears to host system software as two circular queues of message buffers, one for each direction of transfer. Sending a message to the NX/101 host mailbox causes the message to be transferred to the host memory, where it can be read by the host processes. Similarly, receiving a message from the host mailbox causes any messages placed in the host memory by host processes to be transferred to the EXOS/101 process.

Apart from transferring data by means of messages, processes on the EXOS/101 can also directly read and write the the host memory by means of NX/101 calls. The contents of messages sent and received from the host is not interpreted by the NX/101, and is strictly a matter of protocol between the host and the user software.

1.3.6. Ethernet Interface

The Ethernet interface also appears as a special dedicated mailbox. An EXOS/101 process sends a packet over the Ethernet by sending the packet's address in a message to the special mailbox. The packet is formatted according to the Ethernet specifications. The preamble and CRC are generated by the hardware automatically and need not be supplied by the user. After the packet is transmitted a reply message is returned to a user-specified mailbox, returning the packet buffer. Similarly, packets are received from the Ethernet by sending an empty buffer's address in a message to the special mailbox. When the Ethernet controller receives a message, it is stored in the buffer and a reply message is returned to the user-specified mailbox.

Packets arriving over the Ethernet are filtered based on the destination

address. Only those packets whose destination address matches one of addresses specified by the user are received. The address filter is implemented in hardware, but for multicast addresses, it is not perfect. Therefore NX/101 supplements the hardware filter with a somewhat slower software filter which completes the filtering of multicast addresses.

The user specifies receive addresses by means of address slots. Each slot carries one destination address. The user can selectively enable/disable receive on address slots. One address slot is reserved for the physical address and one slot is reserved for the broadcast address. The remaining address slots contain multicast addresses only. The number of multicast address slots is defined by the configuration of the NX/101.

The Ethernet controller can operate in one of several possible modes selectable by the user. Specifically, the user can disconnect the controller from the network, disable/enable the software multicast address filter, enable to receive all packets from the network (promiscuous mode), and reject/accept packets received with errors.

The network management functions are supported by the EXOS/101 by keeping a tally of various events such as the number of packets transmitted/received, packet errors etc. A Time Domain Reflectometry function is also available, to aid system designers and integrators locate faults in the Ethernet cable.

1.3.7. Ethernet Link Level Controller Mode

If the EXOS/101 is to be used in link level controller mode, then most of the description above of NX/101 can be disregarded. In this mode, the host does not download any code to the board. Instead, the host sends command requests to the board which drive the Ethernet interface described above. When a request completes, the EXOS/101 returns a reply message. Transmit and receive commands can be pipelined -- NX/101 uses the 64 Kbytes of dual-ported RAM for buffering packets.

2. REFERENCES

The EXOS/101 conforms to the following specification:

- [1] DEC, Intel, and Xerox Corporations, "The Ethernet: A Local Area Network: Data Link Layer and Physical Layer Specifications," Document no. T588.B/1080/15K, Intel Corp. (September 1980).

The EXOS/101 conforms to the Multibus specification:

- [2] Intel Corp., Intel Multibus Specification, Order Number 9800683-04, Intel Corp., (1982).

The EXOS/101 supports front-end processing of user-written higher-level protocols, on an 8088 CPU:

- [3] Intel Corp., iAPX 88 Book, Document No. C-257/781/75K/RRD, Intel Corp., (1981).

The following reference describes the C language, which is used for procedural specifications in this manual:

- [4] Kernighan, B.W. and Ritchie, D.M, The C Programming Language, Prentice-Hall, Englewood Cliffs, New Jersey (1978).

The following reference describes the ISO Open Systems Model:

- [5] International Organization for Standardization (ISO), "Reference Model of Open Systems Interconnection," Document no. ISO/TC97/SC16 N227 (June 1979).

3. NOTATIONS AND CONVENTIONS

This section describes notations and conventions followed throughout this manual. Any restrictions specified here are applicable to all situations unless otherwise specified. The contents of this section should be carefully read first since the constraints mentioned here will not always be repeated in following sections.

3.1. Number Base

All numbers in this manual are decimal unless postfixed with the letter H, in which case they are hexadecimal.

3.2. Data Object Terminology

The following terms are used to describe data objects of various sizes:

byte:	8 bits
word:	16 bits
longword:	32 bits

3.3. Message Format Specification

The EXOS/101 provides access to some of its services by means of request/reply message pairs. Message formats are specified both in figures and descriptive paragraphs. The figures show the order of data fields, field length, offset from the message beginning, and include a brief description of the field's purpose. Descriptive paragraphs, keyed to the order of fields in the message, provide all necessary details not supplied in the figures.

One column in the message figures, labeled "Request," specifies what value, if any, the field should have in the request message. Another column, labeled "Reply," specifies what value, if any, the reply message returns. When some definite value is specified for a field in a request message, this value must be used, or undefined effects may occur. If a field is designated as "undefined" then it can have any arbitrary value. In the reply message, a field designated as "preserved" will return the same value as was supplied in the original request message. Where more comment is required, the entry "see text," directs the reader to a paragraph labeled with the same index as the field.

3.4. Procedural Specifications

Where it is necessary to describe a procedure, this manual uses the C programming language. Where appropriate, the language has been adapted in the style of pseudo-code. Such departures from the formal specification of C are denoted by enclosure in right-angle brackets, as in this example:

```

init_toxq () {
    for (i=0; i<QLEN; i++) {
        toxq[i].link = <16-bit offset of next buffer address>;
        toxq[i].rsrvd = 0;
        toxq[i].status = TOXINITSTAT;
        toxq[i].length = TOXDATALEN;
        <initialize any user-specified fields>;
    }
}

```

3.5. Bit Position and Value Specifications

When any data object is described in terms of separate bit fields, the Least Significant Bit (LSB) is designated as bit 0 and the Most Significant Bit (MSB) as bit n, where the object's size is n+1 bits. For instance, bit 7 is the MSB of an 8-bit data object.

For programming convenience, bit fields are often described in terms of their OR-maskable numeric value instead of their position, as described above. For instance, if the description of a request mask reads:

```

01  write request bit.
02  read request bit.

```

then a write is specified by bit 0 and a read by bit 1. The value 03 specifies both read and write.

3.6. Data Storage Order

Many applications of the EXOS/101 require the consideration of two different programming environments: that of software on the EXOS/101 itself, and that of software on a host computer which communicates with the EXOS/101. In either environment, it is crucial that user software store data objects which are known to NX/101 firmware in the order which NX/101 expects - and that the programmer understand how NX/101 stores data objects which are known to user software.

In the EXOS/101's own memory address space, NX/101 always interprets data in the 8088 CPU's native order. This means that in any data object of more than one byte, the most significant byte is stored at the higher memory address. For instance, a memory dump of the 32-bit value 0103070FH stored at EXOS/101 memory address 1C83H would appear as follows:

```

1C83: 0F
1C84: 07
1C85: 03
1C86: 01

```

In the Multibus memory address space shared between the EXOS/101 and the host system, NX/101 can interpret data either in the 8088 CPU's native order, or

optionally in the host system CPU's native order. This is controlled by the host data order conversion option, described fully in section 4.2. If the conversion option is not enabled, then any data objects in host memory which NX/101 interprets must appear to the EXOS/101 in the 8088 CPU's native order.

If the conversion option is enabled, then NX/101 will automatically translate between its native order and the host CPU's native order when it reads and writes data to and from the host's memory. It decides what conversions are necessary by examining a constant pattern in host memory at initialization time. Conversions work independently on three data types: byte strings, words, and longwords.

Note that because NX/101 must know the data type to apply the appropriate conversion, the word and longword conversion are applied only to data objects which NX/101 itself interprets, such as configuration information or Ethernet Data Link protocol parameters. Other data objects, such as an Ethernet packet's data field, are subject only to the byte string conversion applied to any data transferred between host memory and EXOS/101 memory.

3.7. Integration with 68000-Based Systems

The host data order conversion for the byte string data type is intended primarily to accommodate microcomputer designs using the 68000 microprocessor (such as those based on the Stanford University Network (SUN) workstation design.) One idiosyncrasy of such processor implementations is that they invert bit 0 of the memory address when performing byte-wide memory operations on the Multibus. This has a more complex effect than a simple byte swap on a word data type. For example, a byte quantity written at logical address 0003H appears at physical address 0002H in Multibus memory. The EXOS/101 automatically compensates for this peculiarity when the host data order conversion option is enabled. It will invert bit 0 of host memory addresses, if required, on all Multibus memory access operations.

3.8. Data Alignment

The EXOS/101 requires special data alignment in only one instance. The configuration message (see section 4.4) must begin on an even logical address in host memory. All data structures defined by NX/101 firmware are designed so that they can be efficiently supported by processors and high-level languages which require even alignment of word and longword data types.

While the EXOS/101 does not generally require even alignment, this practice is still recommended, in order to obtain the best performance from future software-compatible products which utilize a 16-bit data bus.

3.9. Memory Address Format

All memory addresses are 32-bit objects unless otherwise specified. They are stored in memory in the same order as the longword data type. When NX/101's host data order conversion option is enabled, it will apply the same conversions to addresses stored in shared memory.

The interpretation of memory addresses by NX/101 depends on context. Any address which refers to a location in EXOS/101 memory, whether the address

value itself is stored in EXOS/101 memory or in host memory, is always interpreted as a segmented address. This term refers to the 8088 CPU's native address format. A segmented address consists of a 16-bit segment base and a 16-bit offset address. At run time, the 8088 forms a 20-bit absolute address by shifting the segment base left by four places (multiply by 16) and adding the offset to the result. Therefore a segmented address can access 1 Mbyte of memory. Figure 3-1 shows how a segmented address is mapped into the longword data type.

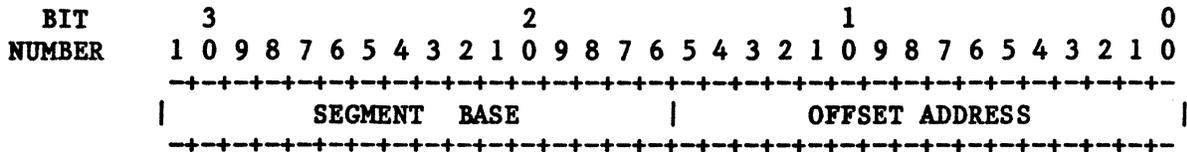


Figure 3-1: Mapping of Segmented Address into Longword Data Type

When a segmented address is stored in EXOS/101 memory, it appears in the following order:

- Byte 0: Offset, low order
- Byte 1: Offset, high order
- Byte 2: Segment, low order
- Byte 3: Segment, high order

Storage order in the host system memory should appear the same to the EXOS/101 unless the host data order conversion option is enabled, in which case it should appear in the host CPU's native order for the longword data type.

The interpretation of addresses which refer to host system memory locations depends on the EXOS/101's host address mode option. In segmented mode, they are interpreted in the same manner as addresses referring to EXOS/101 memory locations. This restricts access to a 1 Mbyte range of host system memory, from 00000H to OFFFFFH. In order to provide access to the full 16 Mbyte Multibus memory address space, the NX/101 also provides an absolute host address mode. An absolute address is a simple 24-bit physical memory address, mapped into the longword data type as shown in figure 3-2.

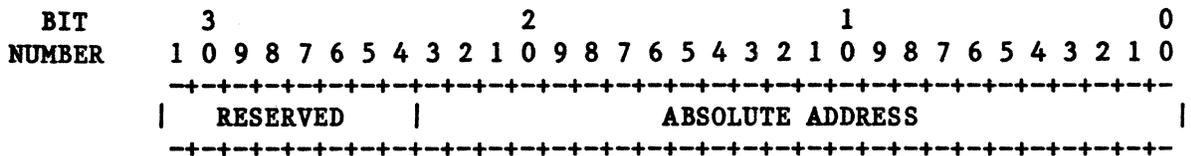


Figure 3-2: Mapping of Absolute Address into Longword Data Type

As shown in the figure, the most significant 8 bits are reserved, and should be set to 0. When an absolute address is stored in EXOS/101 memory, it appears in the following order:

Byte 0: least significant byte
Byte 1: somewhat significant byte
Byte 2: most significant byte
Byte 3: reserved, must be 0

Storage order in the host system memory should appear the same to the EXOS/101 unless the host data order conversion option is enabled, in which case it should appear in the host CPU's native order for the longword data type.

3.10. Shared Multibus Memory Access Restrictions

It is the user's responsibility to ensure that a specified Multibus memory address exists in functional memory. The EXOS/101 does not time out if no memory response is received on the Multibus. To aid diagnostics a Multibus Status LED is provided. Its location on the EXOS/101 is shown in section 11. When this LED is lit, the EXOS/101 is accessing the Multibus. Thus if the LED is constantly lit then most likely the EXOS/101 has been given a non-existent address and is stuck waiting for the response.

The EXOS/101 can access data structures anywhere in the 16 Mbyte Multibus memory space. It accesses this address space by dynamically mapping a half-Mbyte window of its own CPU's address space into Multibus memory. The mapping is at even half-Mbyte boundaries: 0-07FFFFH, 080000H-0FFFFFFH, 100000H-17FFFFH, and so on. User software does not perform either the mapping or the data transfer; it simply gives addresses to NX/101 firmware, which effects the transfer. In general, NX/101 will automatically perform any re-mapping required to transfer data structures which straddle the half-Mbyte bounds. However, for reasons of efficiency, one type of data structure (the host message queues) may not straddle these boundaries. This will be noted in their description in section 4.5.

4. INITIALIZATION AND HOST INTERFACE

This section contains information pertinent to the design of host-resident software, such as an I/O driver, which communicates with the EXOS/101 when it is installed in a Multibus-based system. The host interface can be broken down into two aspects, the initialization procedure, and the communication method used subsequently. Initialization refers to the process which begins upon resetting the EXOS/101, and concludes either with entering the Link Level Controller mode, or with the execution of down-loaded software. During the process of initialization, the host system sets up the host message queue data structures. The host message queue protocol, defined by NX/101 firmware, uses these queues for all further communications between the host processor and the EXOS/101.

The following paragraphs give an overview of the initialization process:

- 1) The host system resets the EXOS/101, which then executes self-diagnostics. If the diagnostics fail, then the EXOS/101 displays an error code on the NX/101 status LED (see section 11) until reset again. If the diagnostics pass, then the EXOS/101 awaits configuration by the host.
- 2) The host system passes the EXOS/101 the address of a configuration message in host memory. The EXOS/101 examines this message, and modifies some fields according to the results of configuration. If configuration is unsuccessful, then the EXOS/101 again displays an error code on the NX/101 status LED until reset. If the configuration message is valid, then the EXOS/101 enters one of three modes, as specified by the message's operation mode field.
- 3) Initialization for each of the three different modes proceeds as follows:
 - a) In Link Level Controller Mode, the EXOS/101 begins to execute firmware which brings NX/101's Ethernet Data Link driver interface out to the host system interface. No software is down-loaded; instead the host system passes Data Link commands to the board, and receives replies, through the standard host message queue protocol. This mode is described fully in section 5.
 - b) In Front-End Mode 1, the host system proceeds to down-load software to the EXOS/101, by passing down-load request messages through the standard host message queue protocol. When the software has been down-loaded, it passes an execute request to the board, which then begins to execute the down-loaded software. Subsequent actions depend entirely on the software which has been installed, although the host message queue protocol remains in place.
 - c) In Front-End Mode 2, the EXOS/101 proceeds to bootstrap itself from the Ethernet interface, as described in section 10. Depending on how the bootstrap server configures the EXOS/101, it may still communicate with the host system through the

standard host message queue protocol. Network bootstrap is quite similar in many ways to initialization by a host processor; the configuration message described in this section is exactly identical.

4.1. Hardware Communications Facilities

Communication between the host processor and the EXOS/101 is conducted via a coordinated exchange of interrupts, I/O instructions, and data transfers through shared memory on the Multibus. The following sections define these primitive channels of communication which are used during the process of initialization and, subsequently, to implement the message queue protocol.

4.1.1. Host Access to the EXOS/101

The host's means of active access to the EXOS/101 are solely through two I/O ports, named port A and port B here for the sake of reference. These ports are accessed over the Multibus, and can be both read and written. Their addresses are selected by jumpers on the EXOS/101, described in section 11.

The effects of reading and writing ports A and B are summarized below:

Read A: resets the EXOS/101 (see section 4.3).

Read B: returns the EXOS/101 status byte:

Bit 0: (Error Bit) when 0, indicates a fatal error in EXOS/101. When the EXOS/101 is reset, this bit is 0, but will be set to 1 if the self test completes successfully. If this bit is not set within 2 seconds, then the EXOS/101 has failed the self diagnostics.

Bit 1: (Interrupt Bit) is set whenever the EXOS/101 asserts a Multibus level interrupt. This is useful when an interrupt line is shared and polling is required. An I/O write to port A clears this bit. The interrupt bit is defined only when level interrupts are selected.

Bit 2: undefined.

Bit 3: (Ready Bit) when 0, indicates that the EXOS/101 is ready to accept a byte written into port B. When 1, the EXOS/101 has not yet read the byte last written into port B.

Bits 4-7: undefined.

Write A: causes the EXOS/101 to drop the interrupt line, when it has asserted a non bus-vectored interrupt on the Multibus. This also clears the interrupt bit in port B. The value written is arbitrary, and is not accessible to software on the EXOS/101.

Write B: interrupts the EXOS/101 CPU, and communicates a 1-byte value. This is the only way to communicate a value to the EXOS/101 other than through shared memory.

4.1.2. EXOS/101 Access to the Host

The EXOS/101 functions as a master on a Multibus system. It can access the full 16-Mbyte memory address space and 64K I/O address space, and interrupt the host processor. User software on the EXOS/101 does not directly control these resources. Instead, it calls NX/101's host interface driver, described in section 8.

In general, data is transferred between the host and the EXOS/101 via shared memory, which may be any portion of system memory accessible to both processors on the Multibus. The EXOS/101's 8088 CPU performs the transfer by dynamically mapping part of its own address space into the Multibus memory address space, and executing a block transfer instruction. Note that the EXOS/101's on-board memory cannot be shared; it is not directly accessible by the host processor.

The EXOS/101 can interrupt the host either through I/O addresses, memory addresses, or the Multibus interrupt lines. The type which will be used is selected at initialization time. Memory and I/O-mapped interrupt addresses are configured by software; the interrupt line is selectable by means of a jumper option, described in section 11. Unless I/O-mapped interrupts are selected, the NX/101 firmware will not normally generate I/O operations on the Multibus. User software on the EXOS/101 can use I/O instructions to control other peripheral cards.

4.2. Host Data Order Conversion Option

The host data order conversion option determines whether the EXOS/101 will interpret data read from host memory according to its own native ordering, or according to the host CPU's native ordering. This option is selected by a field in the configuration message (see section 4.4.5). If enabled, then the NX/101 inspects a known data pattern in the configuration message, written in the host CPU's native order. It determines what conversions are necessary to make this pattern appear in the order it expects, for several different data types: byte array, word array, and longword. NX/101 will then apply the appropriate conversion to all data objects subsequently read from host memory.

For the byte array data type, NX/101 knows how to convert data stored according to the SUN design's byte addressing idiosyncrasies. This means that it will invert the least significant address bit when addressing host system memory, to reverse the effects of common 68000 CPU board designs. For the word data type, NX/101 can swap bytes if necessary. For the longword data type, NX/101 can swap words, swap bytes, or both. Therefore I/O driver software for any reasonably normal host CPU can store data objects in its native order, and leave conversion up to the EXOS/101.

Naturally, the EXOS/101 must know the type of a data object to apply the appropriate conversion. All data objects described in this section are known to NX/101, except for the actual contents of messages between the host and the EXOS/101. NX/101 does apply the byte array conversion (if necessary) to

message contents, and to all data transferred. How the contents of messages should be further interpreted is the function of user-level software running on the EXOS/101. For instance, the firmware which drives the Link Level Controller Mode (see section 5) runs at user level under NX/101, and converts word and longword data objects which are known to itself, but not to NX/101. NX/101 assists this process by providing kernel calls (see section 8.4) which convert word and longword data types as required by the host data order conversion option.

Whether or not the host data order conversion option is enabled, the host system must still write the required data pattern in the configuration message. This pattern occupies 12 bytes of the 32-byte test pattern/memory map field (see section 4.4.10). It should be initialized as shown in figure 4-1. Note that while the relative position of subfields in the test pattern is specified, the order of bytes within those subfields is dependent on the host CPU architecture. Figure 4-2 shows how this pattern might be initialized in the C language, both statically and dynamically.

#	Length	Offset	Sub-Field Name	Value
1)	1	0	Byte 0	01H
2)	1	1	Byte 1	03H
3)	1	2	Byte 2	07H
4)	1	3	Byte 3	0FH
5)	2	4	Word 0	0103H
6)	2	6	Word 1	070FH
7)	4	8	Longword	0103070FH
8)	20	12	: Reserved :	: zero :

|<-----1 byte----->|

Figure 4-1: Host Data Order Conversion Option Test Pattern

Note that memory addresses, regardless of the host address mode, are stored and interpreted as the longword data type. For instance, the longword test pattern can also be regarded as a memory address in the host's native format

for the absolute address 0103070FH (if absolute address mode is selected) or for segment 070FH, offset 0103H (if segmented mode is selected).

If NX/101 cannot make any sense of the test pattern presented by the host, then initialization is aborted, and the appropriate error code displayed on the status LED.

```

/* constants for test pattern */
#define BYTE0 0x01
#define BYTE1 0x03
#define BYTE2 0x07
#define BYTE3 0x0F
#define WORD0 0x0103
#define WORD1 0x070F
#define DWORD 0x0103070F

/* static initialization of test pattern */
struct tstptrn {
    char byteptrn[4];
    short wordptrn[2];
    long lwordptrn;
    char rsvrd[20];
};

struct tstptrn tp = {
    BYTE0, BYTE1, BYTE2, BYTE3,
    WORD0, WORD1,
    DWORD,
    0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
};

/* dynamic initialization of test pattern */
initptrn ()
{
    register int i;
    tp.byteptrn[0] = BYTE0;
    tp.byteptrn[1] = BYTE1;
    tp.byteptrn[2] = BYTE2;
    tp.byteptrn[3] = BYTE3;
    tp.wordptrn[0] = WORD0;
    tp.wordptrn[1] = WORD1;
    tp.lwordptrn = DWORD;
    for (i=0; i<20; i++) tp.rsvrd[i] = 0;
}

```

Figure 4-2: Host Data Format Test Pattern Initialization

4.3. Reset and Configuration Procedure

This section describes initialization by a host system up to the completion of configuration. Figure 4-3 shows a typical procedure which implements as much.

EXOS/101: Initialization and Host Interface

The EXOS/101 is reset by the Multibus INIT signal, or whenever port A is read from the Multibus. Host software should use the latter method to be sure. On reset the EXOS/101 performs a series of self tests to confirm hardware integrity. While these tests run, the NX/101 status LED (see section 11) will remain lit constantly. When self-diagnostics complete successfully, the EXOS/101 sets the error bit in I/O port B and flashes the status LED at regular intervals.

If the error bit is not set within 2 seconds of reset, the host may assume that self-diagnostics turned up a problem. In this case, the EXOS/101 repeatedly reports an error code through the NX/101 status LED (for error code values, see section 11). The EXOS/101 will remain in this state until reset again.

A jumper option, described in section 11, determines how initialization will proceed after reset and self-diagnostics. If the jumper selects network bootstrap, then the EXOS/101 will attempt to down-load software over the Ethernet (see section 10). Otherwise the EXOS/101 awaits configuration by the host processor.

The host configures the EXOS/101 by passing it the address of a configuration message, located in shared memory. This message establishes various NX/101 parameters and selects among several modes of operation. Parameters include memory allocation for NX/101 objects, the address of NX/101's movable data area in EXOS/101 memory, and the location of message queues in shared memory. Among the optional operation modes, the host can select network bootstrap. This will proceed as though the net boot jumper option had been installed, except that NX/101 will first note the contents of the host configuration message. Other configuration options include host data order conversion and the host address mode.

The host processor communicates the address of the configuration message to the EXOS/101 by writing a sequence of 8 bytes into port B. Each byte should be written after checking that the ready bit of the EXOS/101's port B is clear. This ensures that the EXOS/101 is ready to accept the next address byte. The first four bytes of the sequence must be FF-FF-00-00 (sent from left to right). The next four bytes are the configuration message's absolute Multibus memory address (least significant byte first). The configuration message must be aligned on an even address boundary. When the last byte is written, the EXOS/101 reads and interprets the configuration message. If the address for the initialization message is not valid, then the EXOS/101 will display an error code on the status LED (see section 11).

When the EXOS/101 has finished processing the configuration message, it writes a completion code into the appropriate field of the message. Any value other than 0FFH indicates completion; the value 0 indicates successful configuration. Other values denote specific errors in configuration (see section 4.4.3). Normally, configuration should complete within 2 seconds, but network bootstrap might take longer, depending on circumstance. NX/101 also returns a few parameters to the host in the configuration message, notably its version number and a map of available memory.

Once configuration is complete, the memory space occupied by the configuration message can be used for any other purpose. After configuration, communication

```
extern read_port(Port_Num) /* returns value read from port Port_Num */
extern write_port(Port_Num, Val) /* writes Val to port Port_Num */
extern start_clock() /* starts an interval timer */
extern clock() /* returns the current value of the interval timer */

/* bit value definitions for status byte read from port B */
#define ERROR_BIT 1
#define READY_BIT 8
#define ERRNON 0

struct { /* configuration message */
    short reserved;
    char version[4];
    char comp_code;
    <etc...>
} init_msg;

char init_addrs[8] = {0xFF, 0xFF, 0, 0, <absolute address of init msg> };
/* see section 3.9 for absolute address format */

initialize () {

    < set up init_msg and the message queues (see section 4.6) >;

    read_port(A); /* reset the EXOS/101 */

    start_clock(); /* start timer, clock counts real time */

    /* wait until self test completes */
    while ((read_port(B) & ERROR_BIT) == 0 ) {
        if (clock() > 2_SECONDS) {
            return (malfunctioning_board);
        }
    }

    /* write the configuration message address */
    for (i=0; i<8; i++) {
        while ((read_port(B) & READY_BIT) == 1);
        write_port(B,init_addrs[i]);
    }

    /* wait for the reply message */
    while (init_msg.comp_code == 0xFF);

    /* ensure no errors */
    if (init_msg.comp_code != ERRNON)
        return (error);
    else
        return (success);
}
```

Figure 4-3: Typical Reset and Configuration Procedure

between the host and the EXOS/101 is carried out solely by means of message queues, described in section 4.5.

4.4. Configuration Message Format

Figure 4-4 shows the format of the configuration request/reply message. This is used identically by either a host system or a network bootstrap server. The following paragraphs explain the individual fields in detail. Note that reply values other than the completion code field itself are defined only if configuration is successful.

4.4.1. Reserved Field

The first field is reserved for use by NX/101. Its value in the request message must be 1, and its return value is undefined.

4.4.2. EXOS Version Code Field

The EXOS version code field is undefined in the request message. In the reply message, it returns version codes for NX/101 and the EXOS/101 in the form X.Y and A.B, respectively. These are expressed as ASCII digits, one per byte in the order X-Y-A-B, starting from the lower address.

4.4.3. Configuration Completion Code Field

The completion code field must be OFFH in the request message. The EXOS/101 signals that configuration is complete, and returns the completion code, by writing one of the following codes into this field:

00H successful completion.

A4H invalid operation mode.

A5H invalid host data format test pattern. This occurs when NX/101 cannot find any reasonable conversion to derive the expected data pattern from that supplied in the test pattern. In practice, this might imply that the host has given the EXOS/101 the wrong address for the configuration message.

A7H invalid configuration message format. This may occur if reserved fields contain an improper value. In practice, this error message may indicate that the host has given the EXOS/101 the wrong address for the configuration message.

A8H invalid movable block address.

A9H invalid number of processes.

AAH invalid number of mailboxes.

ABH invalid number of address slots.

#	Length	Offset	Field Name	Request	Reply
1)	2	0	Reserved	1	undefined
2)	4	2	EXOS Version Code	undefined	see text
3)	1	6	Configuration Completion Code	OFFH	see text
4)	1	7	EXOS Operation Mode	see text	preserved
5)	2	8	Host Data Format Option	see text	see text
6)	3	10	Reserved	zero	undefined
7)	1	13	Host Address Mode	see text	see text
8)	1	14	Reserved	zero	undefined
9)	1	15	Memory Map Size	zero	see text
10)	32	16	: Test Pattern/Memory Map :	see text	see text
11)	4	48	NX Movable Block Address	see text	see text
12)	1	52	Number of Processes	see text	see text
13)	1	53	Number of Mailboxes	see text	see text
14)	1	54	Number of Multicast Slots	see text	see text
15)	1	55	Number of Hosts	see text	preserved

continued on next page....

Figure 4-4: Configuration Request/Reply Message

#	Length	Offset	Field Name	Request	Reply
....continued from previous page					
16)	4	56	Host-to-EXOS Message Queue Base Address	see text	preserved
17)	2	60	Host-to-EXOS Message Queue Header Address	see text	preserved
18)	1	62	Host-to-EXOS MQ Interrupt Type	see text	preserved
19)	1	63	Host-to-EXOS MQ Int. Value	see text	preserved
20)	4	64	Host-to-EXOS Message Queue Interrupt Address	see text	preserved
21)	4	68	EXOS-to-Host Message Queue Base Address	see text	preserved
22)	2	72	EXOS-to-Host Message Queue Header Address	see text	preserved
23)	1	74	EXOS-to-Host MQ Interrupt Type	see text	preserved
24)	1	75	EXOS-to-Host MQ Int. Value	see text	preserved
25)	4	76	EXOS-to-Host Message Queue Interrupt Address	see text	preserved
<-----1 byte----->					

Figure 4-4a: Configuration Request/Reply Message (continued)

ACH invalid number of hosts.

ADH invalid host message queue parameter. NX/101 returns this error if it detects any inconsistency in the message queue specifications. This might include a bad interrupt type, invalid segment address, bad linking of the message queue buffers, etc.

AEH insufficient memory for movable data block.

AFH net boot failed.

The codes defined above will also be displayed on the status LED if configuration is not successful.

4.4.4. EXOS/101 Operation Mode Field

The EXOS/101 operation mode field determines the mode in which the EXOS/101 is to be used. Three different modes are supported:

- 0 Link Level Controller Mode. This mode brings the Ethernet Data Link interface out to the host interface. No software is down-loaded. It would typically be used when the EXOS/101 is substituted for the traditional non-programmable Ethernet controller board. For details, see section 5.
- 1 Front-End Mode, down-load from the host. In this mode the EXOS/101 is used as a front-end processor. Higher level software is down-loaded by the host.
- 2 Front-End Mode, down-load from the net. In this mode the EXOS/101 is used as a front-end processor and higher level software is down-loaded from the network. For details, see section 10.

All other values for the mode are reserved and their effects are not defined. If the EXOS/101 is already in the process of network bootstrap (meaning that the configuration message has been received from a bootstrap server) then only mode 2 is permitted.

4.4.5. Host Data Order Option Field

The host data order option field enables the host data order conversion option (see section 4.2). Because the byte order of the host CPU will not be known before initialization, this field is actually treated as two one-byte fields. The host should load the same value into each sub-field in the request message. This value is defined bitwise:

Bit 0: Deduce Format Bit. If 0, NX/101 will apply the conversions currently in force. If the board has not been previously configured, then the default conversion will be in force, meaning that no format conversions are applied to data read from the host. If this bit is 1, then NX/101 examines a constant data pattern written by the host in the configuration message's test pattern/memory map field, and deduces what format conversion are necessary to interpret various data types stored in the host CPU's native format.

Bits 1-7: Reserved. These bits must be 0 in the request message.

When initialized, NX/101 examines this field first, and interprets all other fields in the configuration message accordingly. This field is undefined in the reply message.

4.4.6. Reserved Field

This field is reserved for future use. Its value in the request message must be all zeros. Its value in the reply message is undefined.

4.4.7. Host Address Mode Field

The host address mode field determines how NX/101 will interpret addresses which refer to objects in host memory. It is defined bitwise:

Bit 0: Set Mode Bit. If 0, NX/101 will use the address mode currently in force. If the board has not been previously configured, then the default mode will be in force, meaning that NX/101 will interpret all addresses as 8086-style segmented addresses. If this bit is 1, then the next bit determines the new address mode.

Bit 1: Address Mode Bit. The value 0 selects segmented address mode. The value 1 selects absolute address mode.

Bits 2-7: Reserved. These bits must be zero in the request message.

This field is undefined in the reply message.

4.4.8. Reserved Field

This field is reserved for future use. Its value in the request message must be 0. Its value in the reply message is undefined.

4.4.9. Memory Map Size Field

The memory map size field must be 0 in the request message. In the reply message, it returns the number of segments available in EXOS/101 memory for user software. This field contains a valid value only if the EXOS/101 is configured in mode 1 or mode 2.

4.4.10. Test Pattern/Memory Map Field

The test pattern/memory map field serves different purposes in the request and reply messages. In the request message, it must contain the test pattern described in section 4.2, stored in the host CPU's native format.

In the reply message, the test pattern/memory map field contains a map of memory available for user software on the EXOS/101. This map consists of up to 4 segment descriptors, where the actual number is indicated by the last field. Each segment descriptor specifies a memory segment in terms of the lowest address and the highest address included within the segment. Each address is four bytes long, in the segmented format. The lower bound is given first, then the upper bound. This field contains a valid value only if the EXOS/101 is configured in mode 1 or mode 2. If the optional 64K of RAM between 10000H and 1FFFFH is either absent or is malfunctioning, then the map will not contain this segment.

4.4.11. NX/101 Movable Block Address Field

The NX/101 movable block address field can be used to redefine the location of NX/101's movable data area, described in section 6.2. If the EXOS/101 is configured in mode 0, this field must be OFFFFH, OFFFFH. In modes 1 or 2, the value OFFFFH, OFFFFH specifies that the default location be used. If a non-default address is specified, the segment base must be 0. The offset must place the entire block either between 200H and 3FFH, or between 1000H and OFFFFH.

In the reply message, this field returns the actual address of the NX/101 movable data area. The reply value is not defined in mode 0.

4.4.12. Number of Processes Field

The number of processes field configures the maximum number of processes which NX/101 will support. If the EXOS/101 is configured in mode 0, this field must be OFFH. In modes 1 or 2, the value OFFH specifies that the current value be used. The default value, after reset, is 12. Optionally, a value between 1 and 128 can be specified. In the reply message, this field returns the actual number of processes which NX/101 will support. The reply value is not defined in mode 0.

4.4.13. Number of Mailboxes Field

The number of mailboxes field configures the maximum number of mailboxes which NX/101 will support. Note that this number does not include system mailboxes. If the EXOS/101 is configured in mode 0, this field must be OFFH. In modes 1 or 2, the value OFFH specifies that the current value be used. The default value, after reset, is 16. Optionally, a value between 1 and 128 can be specified. In the reply message, this field returns the actual number of mailboxes which NX/101 will support. The reply value is not defined in mode 0.

4.4.14. Number of Multicast Slots Field

The number of multicast slots field configures the maximum number of multicast address slots which NX/101 will support. Note that this number does not include the physical, broadcast, universal, or null slots, which are permanently allocated. If the EXOS/101 is configured in mode 0, this field must be OFFH. In modes 1 or 2, the value OFFH specifies that the current value be used. The default value, after reset, is 8. Optionally, a value between 0 and 252 can be specified. In the reply message, this field returns the actual number of address slots which NX/101 will support. The reply value is not defined in mode 0.

4.4.15. Number of Hosts Field

The number of hosts field specifies the number of host CPUs on the Multibus interface. Permissible values depend on the mode of operation. In all modes, the value OFFH will retain the value currently in force. Upon first configuration, the default value is 1. In operation modes 0 and 1, only the value 1 may otherwise be specified. However in mode 2 (network bootstrap), this field can be either 0 or 1. If 0, then the host message queues are undefined

and the configuration message fields pertaining to them will not be examined. Its value is preserved in the reply message.

4.4.16. Host-to-EXOS Message Queue Base Address Field

The host-to-EXOS message queue base address field specifies the base address of the shared memory which contains the queue data structures for transferring messages from the host to the EXOS/101 (see section 4.5). Addresses for all message queue data structures are 16-bit offsets, calculated relative to this base. NX/101's interpretation of this base address depends on the host address mode selected (see sections 3.9 and 4.4.7).

In segmented mode, this field must contain an 8086-style segmented address, stored according to the convention described for the longword data type (lower-order 16 bits contain the offset, higher-order 16 bits contain the segment). The offset value of this address must be 0; therefore the segment begins on some even 16-byte address boundary. Note that this format is sufficient only to describe a 20-bit address, or 1 Mbyte of host memory.

In absolute mode this field contains a 24-bit absolute memory address, also stored as a longword. The lower-order 24 bits contain the address; the remaining high-order 8 bits are reserved and must be 0. Furthermore, the lower-order 4 bits of the address must also be 0, so that the segment begins on some even 16-byte address boundary. This format can describe 16 Mbytes of host memory.

This field's value is preserved in the reply message.

4.4.17. Host-to-EXOS Message Queue Header Address Field

The host-to-EXOS message queue header address field specifies the offset of the queue header. This offset must be calculated relative to the base address specified for the host-to-EXOS message queue. Its value in the reply message is preserved.

4.4.18. Host-to-EXOS Message Queue Interrupt Type Field

The host-to-EXOS message queue interrupt type field specifies the type of interrupt which the EXOS/101 will use to alert the host of a change in the status of the Host-to-EXOS/101 message queue. Options are:

- 0 no interrupt. The host polls the message queues.
- 1 I/O mapped. The EXOS/101 writes a specified value at the specified I/O port address.
- 2 memory mapped. The EXOS/101 writes a specified value at the specified memory address.
- 3 level interrupt. The EXOS/101 raises one of the Multibus interrupt lines. The line is selectable by jumpers described in section 11. Note that the interrupt remains asserted until the host explicitly clears it, by writing to the EXOS/101's port A (see section 4.1).

If interrupt type 3 is selected, then the EXOS/101 will set the interrupt bit, readable from port B, whenever it asserts a level interrupt. This bit is not defined when other interrupt types are selected.

The value of this field is preserved in the reply message.

4.4.19. Host-to-EXOS Message Queue Interrupt Value Field

The host-to-EXOS message queue interrupt value field is defined only for I/O mapped or memory mapped interrupt types. If these interrupt types are selected, then this value will be written to the specified I/O port or memory address when an interrupt is asserted. The value of this field is preserved in the reply message.

4.4.20. Host-to-EXOS Message Queue Interrupt Address Field

The host-to-EXOS message queue interrupt address field is defined only for I/O mapped or memory mapped interrupt types. If interrupt type 1 is selected, then it contains an 8 or 16-bit Multibus I/O port address in the first word, and the remaining word is undefined. If interrupt type 2 is selected, then it contains a Multibus memory address, which NX/101 will interpret according to the host address mode. The value of this field is preserved in the reply message.

4.4.21. EXOS-to-Host Message Queue Base Address Field

The EXOS-to-host message queue base address field specifies the base address of the shared memory which contains the queue data structures for transferring messages from the EXOS/101 to the host (see section 4.5). This is exactly equivalent to the host-to-EXOS message queue base address field (see section 4.4.16). Its value in the reply message is preserved.

4.4.22. EXOS-to-Host Message Queue Header Address Field

The EXOS-to-host message queue header address field specifies the offset of the queue header. This offset must be calculated relative to the base address specified for the EXOS-to-host message queue. Its value in the reply message is preserved.

4.4.23. EXOS-to-Host Message Queue Interrupt Type Field

The EXOS-to-host message queue interrupt type field specifies the type of interrupt which the EXOS/101 will use to alert the host of a change in the status of the EXOS/101-to-host message queue. Options are:

- 0 no interrupt. The host polls the message queues.
- 1 I/O mapped. The EXOS/101 writes a specified value at the specified I/O port address.
- 2 memory mapped. The EXOS/101 writes a specified value at the specified memory address.

- 3 level interrupt. The EXOS/101 raises one of the Multibus interrupts lines. The line is selectable by jumpers described in section 11. Note that the interrupt remains asserted until the host explicitly clears it, by writing to the EXOS/101's port A (see section 4.1).

If interrupt type 3 is selected, then the EXOS/101 will set the interrupt bit, readable from port B, whenever it asserts a level interrupt. This bit is not defined when other interrupt types are selected. The value of this field is preserved in the reply message.

4.4.24. EXOS-to-Host Message Queue Interrupt Value Field

The EXOS-to-host message queue interrupt value field is defined only for I/O mapped or memory mapped interrupt types. If these interrupt types are selected, then this value will be written to the specified I/O port or memory address when an interrupt is asserted. The value of this field is preserved in the reply message.

4.4.25. EXOS-to-Host Message Queue Interrupt Address Field

The EXOS-to-host message queue interrupt address field is defined only for I/O mapped or memory mapped interrupt types. If interrupt type 1 is selected, then it contains an 8 or 16-bit Multibus I/O port address in the first word, and the remaining word is undefined. If interrupt type 2 is selected, then it contains a Multibus memory address, which NX/101 will interpret according to the host address mode. The value of this field is preserved in the reply message.

4.5. Message Queue Format

Once the EXOS/101 is configured, message queues in shared memory serve all further communications with the host. This includes software download, packet exchange, and link level controller functions. Two message queues are maintained by the NX/101 firmware, one for each direction of transfer. This section describes the format of the data structures which compose a message queue. Following sections describe how these must be initialized, and then the protocol which ensues after configuration.

Each message queue necessarily includes one queue header and a singly-linked, circular list of message buffers. The required queue header belongs to the EXOS/101; typically the host will maintain its own, separate queue header for each queue. The EXOS/101 queue header and all message buffers must lie within a single 64K area of memory, called the queue segment. Furthermore, each queue must lie entirely within the half-Mbyte mapping boundaries described in section 3.10.

Message queue data structures are described here as viewed by NX/101. The configuration message provides NX/101 with the queue segment base and the offset address of the queue header, for each queue. NX/101 regards all address fields in the queue data structures as 16-bit offsets calculated relative to the queue segment base. As long as this view is preserved for NX/101, users are perfectly free to augment these data structures in any manner necessary to implement the desired abstractions for the host software.

Figure 4-5 shows the format of a message buffer, and the following paragraphs describe the individual fields in detail.

#	Length	Offset	Field Name
1)	2	0	Link
2)	1	2	Reserved
3)	1	3	Status
4)	2	4	Length
5)	n	6	: Data : : : : :

|<-----1 byte----->|

Figure 4-5: Message Buffer Format

4.5.1. Link Field

The link field is the address of the next buffer in the circular queue. This address must be an offset calculated relative to the queue segment base specified in the configuration message. This field is static and should not be changed after configuration.

4.5.2. Reserved Field

This field is reserved. It must be initialized with the value 0.

4.5.3. Status Field

The status field is used to implement the message protocol, and is defined bit by bit:

- Bit 0: Owner bit. If 0 then the buffer is owned by the host; if 1 then the buffer is owned by the EXOS/101. The host may alter a message buffer only while it has ownership.
- Bit 1: Done bit. The EXOS/101 sets this to 0 along with the owner bit every time it passes a buffer to the host. Host software can use the done bit to distinguish between buffers newly received from the EXOS/101 and buffers it has already processed.

Bit 2: **Overflow Bit.** The EXOS/101 sets this bit to 1 if the message had to be truncated because the host receive buffer was shorter than the message sent.

Bits 3-7: undefined. These bits are reserved for the EXOS/101, and should not be used for any purpose by the host.

4.5.4. Length Field

The length field specifies the number of bytes in the data field. The maximum length of the data field is a matter of agreement between the host and the user software on the EXOS/101. There is no restriction on the size of the data field as long as the buffers satisfy the queue segment constraints. Most applications will transfer small amounts of control information via messages, and use direct memory access to move larger data buffers.

4.5.5. Data Field

The data field contains the actual message data passed between the host and the EXOS/101. NX/101 does not interpret its contents in any way - it is exactly equivalent to the data field in messages as seen by processes on the EXOS/101 (see section 8). However, if the host data order conversion option is enabled, and SUN-style address bit inversion is required, this conversion will be applied to the contents of the data field.

4.6. Message Queue Initialization

The host must initialize the message queues and the queue headers prior to configuring the EXOS/101. Figure 4-6 shows the relation between queue headers and message queue buffers at initialization time for a typical implementation. In each queue, the host and EXOS/101 queue headers should point to the same buffer.

For each queue, the link fields should be initialized to form a circular, singly-linked list. This ring structure should not be modified after configuration. Each queue may contain an arbitrary number of buffers, so long as at least one is supplied. The reserved field of all message buffers in both queues should be set to 0.

In the host-to-EXOS queue the status field of all buffers should contain the value 02H, which indicates that they are owned by the host. The length and data fields are not defined at initialization.

In the EXOS-to-host queue the status field of all buffers should contain the value 03H, which indicates that they are owned by the EXOS/101. The length field of each buffer should not exceed the size of the data buffer. Note that the length field must be initialized to accommodate the length of the largest message expected from the EXOS/101, or the message will be truncated upon reception. The data field is not defined at initialization.

Figure 4-7 is a snapshot of an example EXOS-to-host message buffer queue at the time of initialization. This example assumes an 8086-based host system, where the EXOS/101 is configured in the segmented host address mode. The configuration message describing the queue is also shown in part. Data

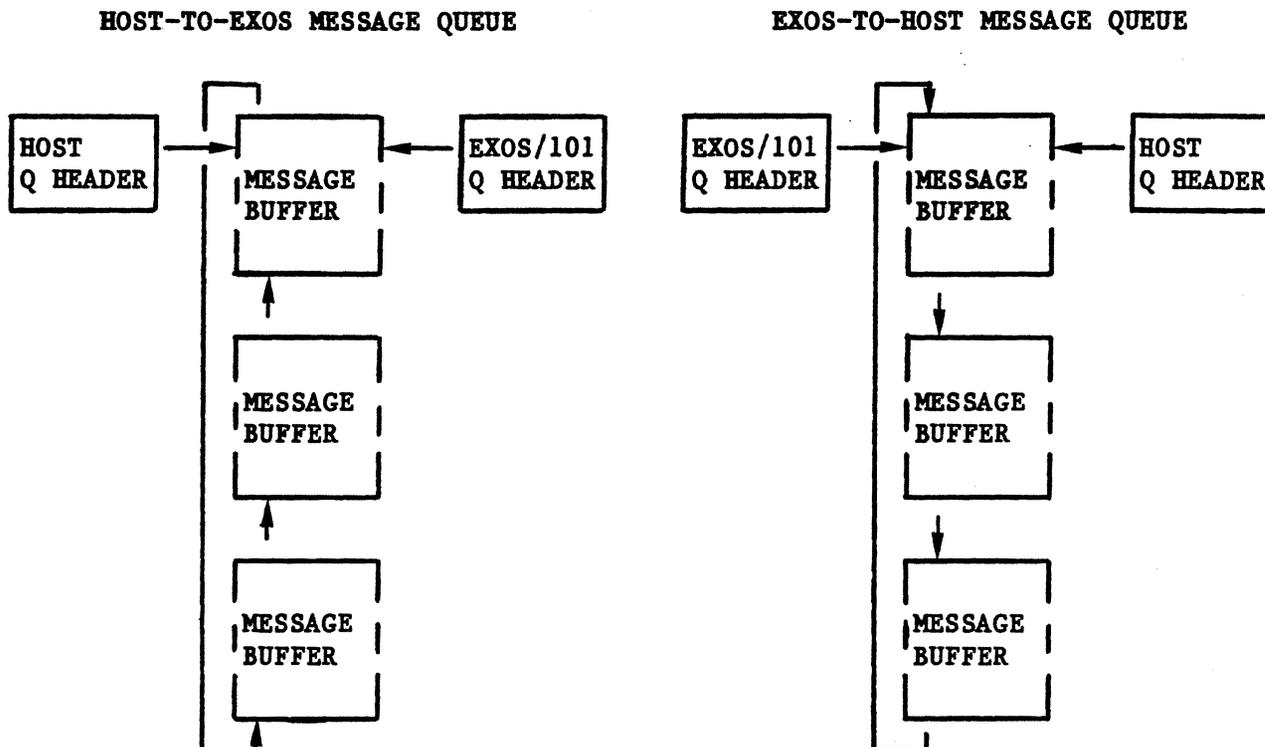


Figure 4-6: Message Queue Data Structures at Initialization Time

structures are shown as vectors containing hexadecimal byte values. The Multibus physical address of each data structure is shown to the left (slightly above the location), and its name to the right. According to the configuration message in this example, writing the value 40H at memory location 0E2044H will interrupt the host. NX/101 will assert this interrupt when the status of the EXOS-to-host message queue changes, as described in the following section. The circular message queue shown here contains three buffers of equal length, each providing a 32-byte data field. The queue header points to one of the buffers, arbitrarily chosen, at its link field address.

4.7. Message Queue Protocol

This section describes the protocol which NX/101 follows in sending messages to, and receiving messages from, the host processor. As it happens, host software can follow the same procedure, so that the exchange is symmetrically defined. The description below assumes such an implementation, but certainly other methods are possible, within the constraints of NX/101's behavior.

In a typical implementation, the host system and the EXOS/101 each maintain private queue headers for both queues. The EXOS/101's host-to-EXOS message queue's header points to the message buffer which NX/101 will receive next. The EXOS/101's EXOS-to-host message queue's header points to the message buffer which NX/101 will send to next. NX/101 maintains these queue headers after configuration. Although the EXOS/101 queue headers are kept in host

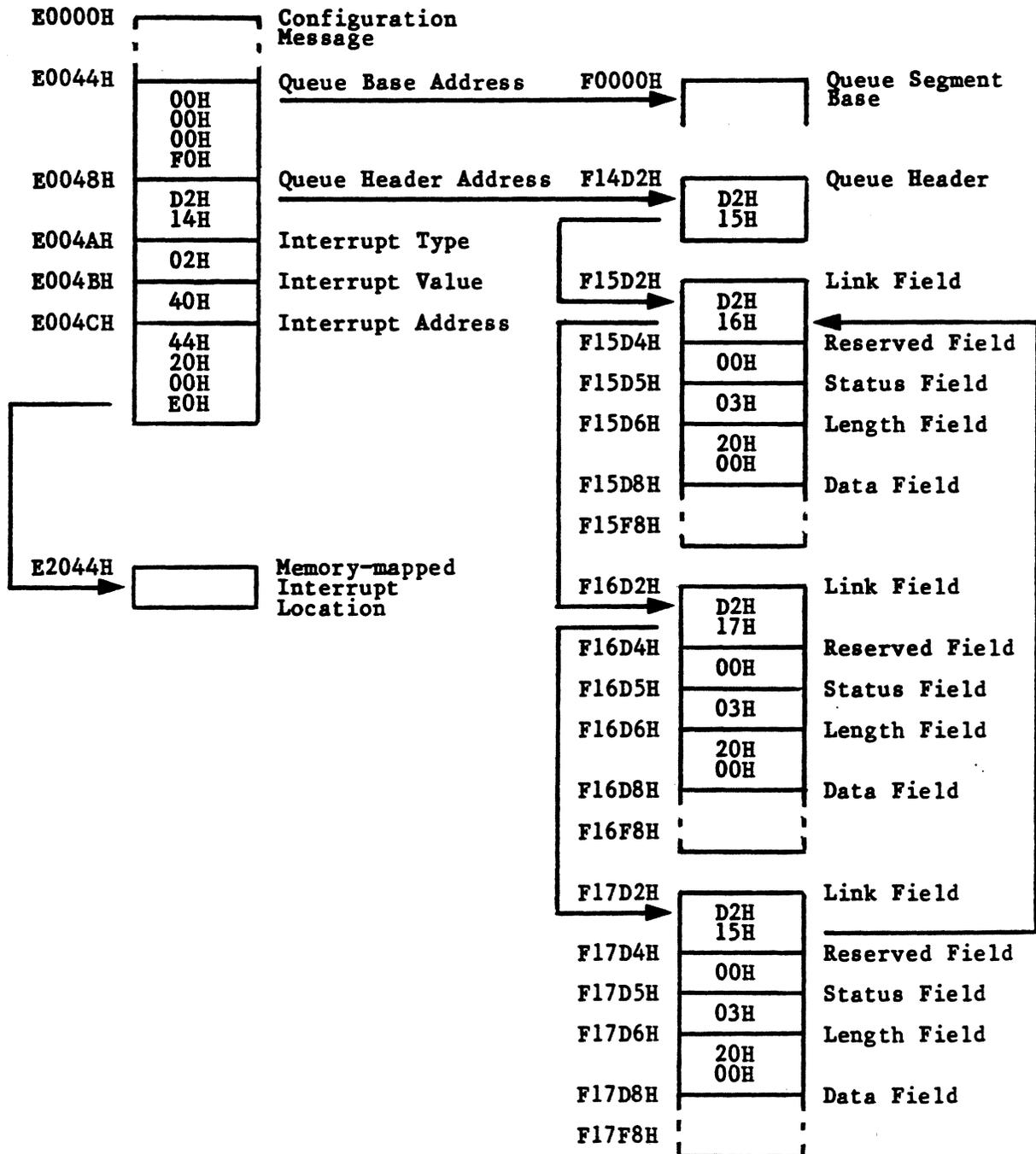


Figure 4-7: Example EXOS-to-Host Message Queue, at Initialization

memory, after initialization the host should not refer to these. Similarly, the EXOS/101 will not refer to the host's own queue headers.

For the host-to-EXOS queue, the host's queue header should always point to the next buffer in which the host will send a message. The EXOS/101's queue header will always point to the next buffer in which the EXOS/101 will look for a message. Both pointers will always move sequentially through the message queue. During the course of message processing, the host's queue header may end up several buffers ahead of the EXOS/101's queue header, but should never "lap" it from behind. Any difference between the headers represents buffers which the EXOS/101 has not yet consumed.

For the EXOS-to-host queue, the host's queue header should always point to the next buffer in which the host will look for a message. The EXOS/101's queue header will always point to the next buffer in which the EXOS/101 will send a message. As above, both pointers will always move sequentially through the message queue. During the course of message processing, the EXOS/101's queue header may end up several buffers ahead of the host's queue header, but again, should never "lap" it from behind. Any difference between the headers represents buffers which the host has not yet consumed.

4.7.1. Host-to-EXOS Message Transfer

Host software can use the following sequence of steps to transfer messages to the EXOS/101:

- 1) Test the owner bit of the buffer to which the host queue header points. If the EXOS/101 still owns this buffer, then wait until it is returned (either poll the owner bit, or wait for the interrupt which accompanies each buffer turnover event).
- 2) Load the link field of the current buffer into the queue header, so that it now points to the next buffer in the queue.
- 3) Load the message into the data field of the current buffer, and set the length field appropriately.
- 4) Change the current buffer's owner bit, so that the buffer now belongs to the EXOS/101.
- 5) Interrupt the EXOS/101 by writing to port B, to notify it that a new message is available.

The EXOS/101 can process more than one message from the host upon receiving a single interrupt. Therefore it is important that the host change the buffer's owner bit only after preparing the other fields. Otherwise, if the EXOS/101 is still processing a previous interrupt from the host, it may consume a half-baked message. Note that the host may prepare more than one message buffer at a time, and send a single interrupt, if sufficient buffers are available.

When the EXOS/101 receives an interrupt from the host, it will:

EXOS/101: Initialization and Host Interface

- 1) Examine the owner bit of the buffer to which the EXOS/101 queue header points. If the buffer belongs to the EXOS/101, then it will process it, as described in the following steps. (Otherwise, the interrupt could mean that the host is returning an EXOS-to-host message buffer, or could be spurious.)
- 2) Load the link field of the current buffer into the queue header, so that it now points to the next buffer in the queue.
- 3) Extract the message from the current buffer. If there is no consumer for this data (no receive request on the NX/101's host interface mailbox), then wait.
- 4) Turn over the current buffer's owner bit, so that the buffer is returned to the host. Set the buffer's done bit to 0.
- 5) Interrupt the host to notify it that a buffer has been returned. The type of interrupt is specified by the configuration message. Repeat from step 1, until the owner bit shows that no new messages are pending.

Note that the interrupt described in step 5 is the same interrupt which the host waits upon when no message buffers are available.

4.7.2. EXOS-to-Host Message Transfer

When the EXOS/101 has a message to transfer to the host, NX/101 will:

- 1) Test the owner bit of the buffer to which the EXOS/101 queue header points. If the buffer belongs to the EXOS/101, then process it, as described in the following steps. Otherwise, wait for an interrupt from the host which indicates that a buffer has been returned (NX/101 can process other jobs in the mean time).
- 2) Load the link field of the current buffer into the queue header, so that it now points to the next buffer in the queue.
- 3) Load the message into the data field of the current buffer, and set the length field appropriately.
- 4) Change the current buffer's owner bit, so that the buffer now belongs to the host. Set the buffer's done bit to 0.
- 5) Interrupt the host to notify it that a new message is available. The type of interrupt is specified by the configuration message.

When the host receives an interrupt from the EXOS/101, it can:

- 1) Examine the owner bit of the buffer to which the host queue header points. If the buffer belongs to the host, then it should process it, as described in the following steps. (Otherwise, the interrupt could mean that the EXOS/101 is returning a host-to-EXOS message buffer, or could be spurious.)

EXOS/101: Initialization and Host Interface

- 2) Load the link field of the current buffer into the queue header, so that it now points to the next buffer in the queue.
- 3) Extract the message from the current buffer. If there is no consumer for this data, then wait.
- 4) Turn over the current buffer's owner bit, so that the buffer is returned to the EXOS/101.
- 5) Interrupt the EXOS/101 by writing to port B, to notify it that a message buffer has been returned. Repeat from step 1, until the owner bit shows that no new messages are pending.

Note that whenever the host receives a non bus-vectored interrupt from the EXOS/101, it should write to the EXOS/101's port A before processing any message queue events. This causes the EXOS/101 to drop its interrupt line, permitting the host to recognize another interrupt. During the host's interrupt service routine, it is assumed that further interrupts from the EXOS/101 are disabled, but that the host's interrupt controller will still buffer one interrupt from the EXOS/101 until leaving the service routine and re-enabling interrupts at that level.

The EXOS/101 will assert an interrupt whether or not the host has cleared its interrupt line - therefore interrupts may merge together, so far as the host can tell. This is why the host should be prepared, whenever it receives an interrupt, to process multiple messages and/or buffers returned by the EXOS/101. Furthermore, the host should be prepared to receive a spurious interrupt from the EXOS/101.

Although the above description assumes that the EXOS/101 is programmed to interrupt the host to signal message queue events, the host also has the option of simply polling the message queue.

4.8. Down-Loading Software from the Host

Normally, if the EXOS/101 is configured in mode 1, host software would then download and run higher level protocol software. Two message formats are provided for this purpose, one to copy user code and data to the EXOS/101, and another to start code execution. For each message the EXOS/101 sends a corresponding reply message which confirms the completion of the request.

4.8.1. Host Down-Load Request

The host can copy code to any location in EXOS/101 memory which is normally available to the user. The down-load request copies buffers up to 64K-1 each in size, in any order, without modification. NX/101 does not protect the user area against un-intentional overlays. Figure 4-8 shows the format of the down-load request/reply message, and the following paragraphs describe the individual fields in detail.

4.8.1.1. Reserved Field

The first field is reserved for use by NX/101, and must be set to 0. Its value in the reply message is undefined.

#	Length	Offset	Field Name	Request	Reply
1)	2	0	Reserved for NX Usage	zero	undefined
2)	4	2	User Id Code	undefined	preserved
3)	1	6	Request Code	00H	preserved
4)	1	7	Return Code	undefined	see text
5)	2	8	Data Length	see text	see text
6)	4	10	Source Address	see text	undefined
7)	4	14	Destination Address	see text	undefined

|<-----1 byte----->|

Figure 4-8: EXOS/101 Down-Load Request/Reply Message

4.8.1.2. User Id Code Field

The user id code field is not interpreted by the EXOS/101, and is returned unmodified in the reply message. It can be used to establish a correspondence between request and reply messages.

4.8.1.3. Request Code Field

The request code field defines the request. Its value in the request message must be 0. This value is preserved in the reply message.

4.8.1.4. Return Code Field

The reply code field is undefined in the request message. In the reply message, it reports the status of the down-load request:

0 successful completion.

A3H destination memory block overlaps the memory reserved for NX/101, no copy done.

AlH invalid request, the EXOS/101 is not in front end mode.

4.8.1.5. Data Length Field

The data length field specifies the number of bytes to be copied into EXOS/101 memory. This may be any value between 0 and 64K-1. In the reply message, this field returns the number of bytes actually copied.

4.8.1.6. Source Address Field

The source address field specifies the starting address in shared memory from which to copy the user code image. This may be either a segmented or an absolute address, depending on the host address mode option. Its value in the reply message is undefined.

4.8.1.7. Destination Address Field

The destination address field specifies the starting address in EXOS/101 memory to which the user code image will be copied. This must be a segmented address. Its value in the reply message is undefined.

4.8.2. Start Execution Request

#	Length	Offset	Field Name	Request	Reply
1)	2	0	Reserved for NX Usage	zero	undefined
2)	4	2	User Id Code	undefined	preserved
3)	1	6	Request Code	00H	preserved
4)	1	7	Return Code	undefined	see text
5)	4	8	Starting Address	see text	preserved

|<-----1 byte----->|

Figure 4-9: EXOS/101 Start-Execution Request/Reply Message

After down-loading protocol software, the host processor starts it executing with a single start execution request message. Once this command has been issued and the reply received, the EXOS/101 does not itself process any more messages. Instead, all messages sent to the EXOS/101 will be queued up for user processes running under the NX/101 kernel.

The start execution request specifies the location at which execution of user code begins. User code is entered as a single process with priority 255 and infinite time slice. All registers except for the PC and stack pointer are undefined. The initial process stack is provided from the NX/101 data area and is guaranteed to be at least 100H bytes deep. The process is free to switch to a bigger stack if desired. In all other respects, it is a normal process, as defined in section 6.4.

Figure 4-9 shows the format of the start execution request/reply message, and the following paragraphs describe the individual fields in detail.

4.8.2.1. Reserved Field

The first field is reserved for use by NX/101, and must be initialized as 0. Its value in the reply message is undefined.

4.8.2.2. User Id Code Field

The user id code field is not interpreted by the EXOS/101, and is returned unmodified in the reply message. It can be used to establish a correspondence between request and reply messages.

4.8.2.3. Request Code Field

The request code field defines the request. Its value in the request message must be 2. This value is preserved in the reply message.

4.8.2.4. Return Code Field

The reply code field is undefined in the request message. In the reply message, it reports the status of the start execution request.

0 successful completion.

A2H invalid starting address, execution not started.

A1H invalid request, the EXOS/101 is not in front end mode.

4.8.2.5. Starting Address Field

The starting address field specifies the initial value of the initial process's program counter. This must be a segmented address. Its value is preserved in the reply message.

5. LINK LEVEL CONTROLLER MODE

In the link level controller mode, the EXOS/101 provides a standard Ethernet Data Link interface to the host system. The host system selects link level controller mode at initialization time, by specifying operation mode 0 in the configuration message (see section 4.4.4). The host does not then download software; instead the EXOS/101 runs firmware which brings NX/101's on-board Ethernet driver out to the host interface. The host can then access all Ethernet functions by exchanging request and reply messages with the EXOS/101 via the message protocol described in section 4.5. The EXOS/101 uses its RAM primarily to buffer packets in both directions between the network and the host.

Link level controller mode functionality is very similar to the NX/101 Ethernet interface for EXOS/101-resident software, described in section 7. Because the underlying objects and capabilities of this mode are identical, they will not be described here in the same detail. Instead, this section concentrates on the format and usage of request messages.

5.1. The Controller Mode Interface

After the EXOS/101 has been initialized in mode 0, the host sends commands as request messages in the host-to-EXOS queue. When a request is completed, the EXOS/101 places a reply message in the EXOS-to-host queue. These queues may be arbitrarily long, and can be used to pipeline Ethernet operations. Figure 5-1 shows how messages are encapsulated in the message queue buffers.

In link level controller mode, the EXOS/101 honors six request messages:

TRANSMIT	send packet from host memory onto Ethernet
RECEIVE	receive packet from Ethernet into host memory
NET_MODE	read/modify the net mode
NET_ADDR	read/modify an address slot
NET_RECV	enable/disable receive on an address slot
NET_STSTCS	read/clear the network statistics

The first two requests above correspond to the transmit and receive messages which on-board software would send to the Ethernet system process under NX/101 (see sections 7.1 and 7.2). The latter four requests correspond exactly to the NX/101 calls by the same name which on-board software would use (see section 9).

Figure 5-2 shows conceptually how requests are processed by the EXOS/101. According to the message queue protocol, as soon as the host software has placed a request message in a host-to-EXOS message queue buffer, it interrupts the EXOS/101. When interrupted, the EXOS/101 reads the requests from the queue and buffers them in its on-board memory.

A request is said to be outstanding once it has been read from the host request queue, and until the corresponding reply message has been written to the host reply queue. The EXOS/101 can buffer up to 32 outstanding request messages. Additional requests will remain in the host request queue until buffers are made available by request completion in the EXOS/101. This should

REQUEST/REPLY MESSAGE BUFFER

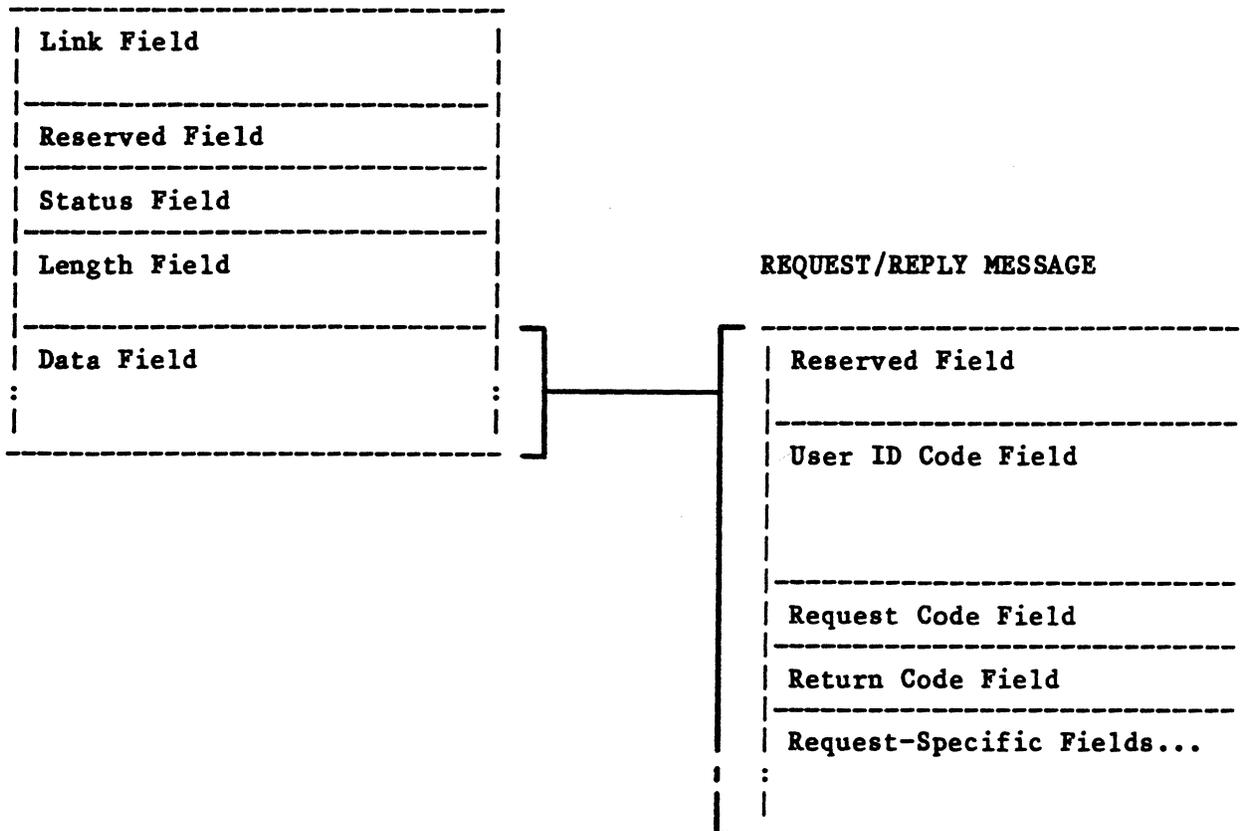


Figure 5-1: Encapsulation of Request/Reply Message in Message Buffer

be noted when designing host software, since certain implementations could become deadlocked by outstanding requests. In particular, receive requests remain outstanding at least until a packet is received from the network. In general, no more than 32 receive requests should be made at any time. Note that in link level controller mode, the EXOS/101 will buffer incoming packets (that pass address filtering) even if no receive requests have been submitted.

As shown by figure 5-2, the EXOS/101 effectively places different request messages in separate internal queues and processes them asynchronously, according to their type. Network management requests are generally processed immediately, and transmit requests are processed as fast as the Ethernet Data Link protocol permits. Receive requests remain outstanding until packets arrive on the Ethernet, unless received packets are already buffered up in the EXOS/101.

The EXOS/101 sends reply messages back to the host immediately upon request completion, which is not necessarily the order in which they are accepted. In order to ensure a certain sequence of operations among requests of different types, a request should be issued only after the reply message for the preceding operation in the sequence has been received. Each request message carries

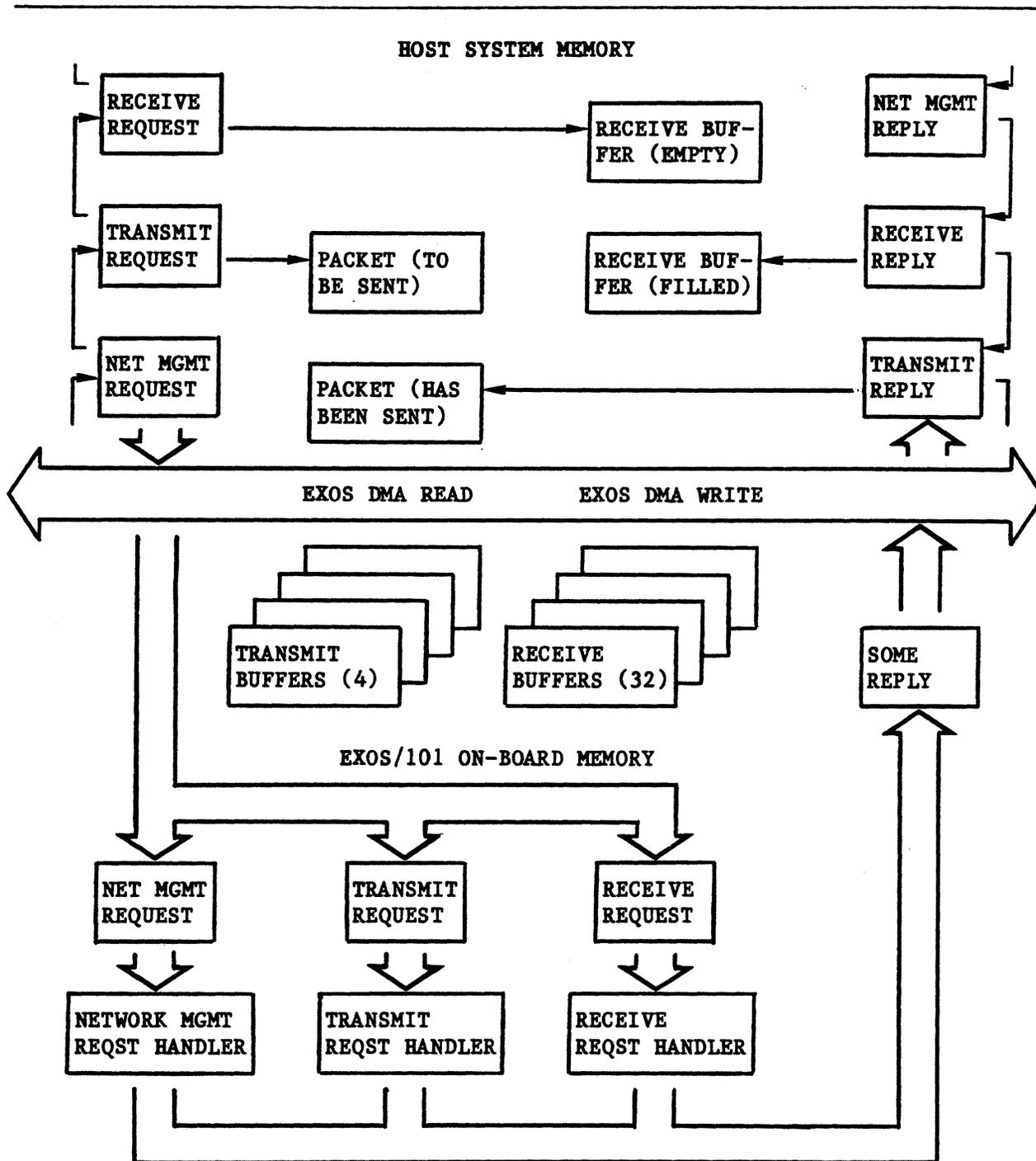


Figure 5-2: Link Level Controller Mode Request Processing Scheme

a 32-bit user id field which is not interpreted by the EXOS/101 and which is returned unmodified in the reply message. This field can be used for any purpose, for example, to establish a correspondence between a request and its reply message.

EXOS/101: Link Level Controller Mode

The remainder of this section specifies the format of the request/reply messages for each request. Where these requests map directly into NX/101 calls (see section 9), the figures also mention the corresponding CPU registers, if any, in parentheses (request, reply).

In addition to the error codes defined for NX/101 calls, any request may return the general error code 0A1H if (a) the request message is shorter than the specified length, (b) an invalid request code is used, or (c) the EXOS/101 is not initialized in link level controller mode.

5.2. TRANSMIT Request/Reply Message

#	Length	Offset	Field Name	Request	Reply
1)	2	0	Reserved for NX Usage	zero	undefined
2)	4	2	User Id Code	undefined	preserved
3)	1	6	Request Code	see text	preserved
4)	1	7	Return Code	undefined	see text
5)	1	8	Address Slot	undefined	see text
6)	1	9	Number of Data Blocks	see text	preserved
7)	2	10	Data Block Length	see text	preserved
8)	4	12	Data Block Address	see text	preserved
n)			: (The two fields above may : appear up to eight times, as : specified by the Number of Data Blocks parameter)		
<-----1 byte----->					

Figure 5-3: TRANSMIT Request/Reply Message

To transmit a packet on the Ethernet, host software sends a transmit request message to the EXOS/101. This message contains pointers to an Ethernet packet in host memory. Packets are prepared for transmission in standard Ethernet data link layer frame format, as described in section 7.1. Host software should prepare the address and type fields. Packets should not include preamble or CRC fields, which are prepared by EXOS/101 hardware. If it serves the purposes of host software, the packet may be composed of up to eight disjoint blocks in host memory.

The EXOS/101 enqueues transmit requests, and completes packet transmission without any intervention from the host. When the EXOS/101 accepts a transmit request, it gathers the packet (or packet fragments) from host memory, and

assembles the packet in an internal transmission buffer. Four such buffers are allocated in link level controller mode, and transmission requests are pipelined - if more than four transmit requests are pending, the packet is not necessarily read from host memory immediately upon acceptance of a new request. This is unlikely, unless the network is very heavily loaded.

If the EXOS/101 is in off net mode (described in section 7.3) then transmit requests will be enqueued, but will remain outstanding until the EXOS/101 is put back in an on net mode. If the EXOS/101 is taken off net during a transmission, then the current transmission will first be completed. If the net disable option is selected (see section 7.4), then transmission will appear to complete normally, but nothing is actually sent on the Ethernet.

An alternate form of the transmit request is provided in link level controller mode only. This is transmit with self-receive, and is selected by the request code OEH (instead of OCH). When this form of the transmit request is used, transmission occurs just as with a normal transmit request, but also generates a received packet - if the destination address passes the established address filtering. Address filtering is performed according to normal procedure for incoming packets with one difference: in imperfect filtering mode, multicast packets are always self-received.

Transmit requests are dispatched in the order they are received from the host system. When the request is completed, the EXOS/101 modifies the request message according to the status of the transmission and returns it to the host as a reply message. Until the reply message is received through the EXOS-to-host message queue, the indicated Ethernet packet belongs to the EXOS/101 and should not be modified.

Figure 5-3 shows the format of the Ethernet transmit request/reply message, and the following paragraphs describe its individual fields in detail.

5.2.1. Reserved Field

The first field is reserved for use by NX/101, and must be set to 0. Its value in the reply message is undefined.

5.2.2. User Id Code Field

The user id code field is not interpreted by the EXOS/101, and is returned unmodified in the reply message. It can be used to establish a correspondence between request and reply messages.

5.2.3. Request Code Field

The request code field defines the request:

OCH transmit.

OEH transmit with self-receive.

This field's value is preserved in the reply message.

5.2.4. Return Code Field

The return code field value is undefined in the request message. In the reply message, it reports the status of the transmission request:

- 00H successful transmission, no retry.
- 01H successful transmission, 1 retry.
- 02H successful transmission, more than 1 retry.
- 10H transmission failed, excessive collisions.
- 40H transmission failed, transmit length not in range.
- 0A1H failed, the EXOS/101 is not in controller mode.

5.2.5. Address Slot Field

The address slot field is an index into the address slot array. Its value in the request message is undefined. In the reply message, it contains the address slot number by which this packet would be received by this station. For instance, the value 255 indicates that the packet was broadcasted, and should be auto-received. Or, if the packet was transmitted to this station's own address, the value would be 253. A zero value means that no slot matched - this packet would not be received by this station.

5.2.6. Number of Data Blocks Field

The number of data blocks field specifies the number of data length/data address field pairs that follow this field in the request message. Each pair describes one block, where a packet may occupy up to eight disjoint blocks in shared memory. This field's value is preserved in the reply message.

5.2.7. Data Block Length Field

The data block length field specifies the length in bytes of the data block whose address follows. The sum of all data block length fields should be the total packet length. This value does not include the preamble or CRC fields, which are appended by EXOS/101 hardware. In the reply message, this field's value is preserved.

5.2.8. Data Block Address Field

The data address field specifies the address of a data block in shared memory, where up to eight blocks compose a packet. Note that the packet, as handed over to the EXOS/101, does not include a preamble, so that the address of the first block will point to the first byte of the packet's destination field. The data address field is preserved in the reply message.

5.3. RECEIVE Request/Reply Message

#	Length	Offset	Field Name	Request	Reply
1)	2	0	Reserved for NX Usage	zero	undefined
2)	4	2	User Id Code	undefined	preserved
3)	1	6	Request Code	ODH	preserved
4)	1	7	Return Code	undefined	see text
5)	1	8	Address Slot	undefined	see text
6)	1	9	Number of Buffer Blocks	see text	preserved
7)	2	10	Buffer Block Length	see text	see text
8)	4	12	Buffer Block Address	see text	preserved
n)			: (The two fields above may : appear up to eight times, as : specified by the Number of Buffer Blocks parameter)		
<-----1 byte----->					

Figure 5-4: RECEIVE Request/Reply Message

Host software receives a packet on the Ethernet by sending an Ethernet receive request message to the EXOS/101. This message contains pointers to a packet buffer in host memory. If the EXOS/101 has already received a packet from the Ethernet, then it will copy the packet into the host buffer. Otherwise the request will not complete until a packet is received.

Received packets are returned to the host in standard Ethernet data link layer frame format, as described in section 7.1. Address, type, and CRC fields are included, but not the preamble. The EXOS/101 performs address and CRC checks in hardware. If it serves the purposes of host software, the packet buffer may comprise up to eight disjoint blocks in host memory.

The EXOS/101 will receive packets from the Ethernet according to several criteria. One is the mode of operation, which determines whether to listen at all, and which categories of address to accept. Another factor is the address filter, which determines the physical address, and up to 252 active multicast addresses. The last factor to consider is the options mask, which defines acceptable errors in received packets. Subsequent sections describe these factors in more detail.

When a packet on the Ethernet satisfies the criteria for reception, the EXOS/101 receives and buffers the packet in its own memory. In link level controller mode, EXOS/101 provides 32 full-size on-board packet buffers which are chained in controller hardware. Therefore it can receive 32 Ethernet packets back-to-back, with minimal interframe spacing, even when no receive requests from the host are pending.

When reception is complete, the EXOS/101 modifies the request message according to the status of the reception and returns it as a reply message. Receive requests are queued up and dispatched in the order received. Until the reply message is received through the host-to-EXOS message queue, the indicated buffer belongs to the EXOS/101 and should not be used.

Figure 5-4 shows the format of the Ethernet receive request/reply message, and the following paragraphs describe its individual fields in detail.

5.3.1. Reserved Field

The first field is reserved for use by NX/101, and must be set to 0. Its value in the reply message is undefined.

5.3.2. User Id Code Field

The user id code field is not interpreted by the EXOS/101, and is returned unmodified in the reply message. It can be used to establish a correspondence between request and reply messages.

5.3.3. Request Code Field

The request code field defines the request. Its value in the Ethernet receive request message must be 0DH. This value is preserved in the reply message.

5.3.4. Return Code Field

The return code field value is undefined in the request message. In the reply message, it reports the status of the receive request:

- 00H packet received with no error.
- 04H packet received longer than buffer supplied, truncated.
- 10H packet received with alignment error.
- 20H packet received with CRC error.

40H no packet received, buffer supplied was less than 64 bytes.

0A1H failed, the EXOS/101 is not in controller mode.

Note that packets with errors are actually received only if the network mode is set appropriately.

5.3.5. Address Slot Field

The address slot field is an index into the address slot array. Its value in the request message is undefined. In the reply message, it contains the address slot number which matched the destination address of the packet received. If the controller is in promiscuous mode, then this field will return the universal address slot, whether or not any address matched. If the controller is not in perfect filtering mode, then this field will return the universal address slot when any multicast packet is received.

5.3.6. Number of Buffer Blocks Field

The number of buffer blocks field specifies the number of buffer length/buffer address field pairs that follow this field in the request message. Each pair describes one block, where a buffer may consist of up to eight disjoint blocks in shared memory. This field's value is preserved in the reply message.

5.3.7. Buffer Block Length Field

The buffer block length field specifies the length in bytes of the buffer block whose address follows. The sum of all buffer block length fields should be the total packet length. The length does not include the preamble but must include 4 bytes for the frame check sequence (CRC) field. In order to receive the longest possible Ethernet packet, the buffer must be at least 1520 bytes long. Minimum size is 64 bytes, which will fit the shortest possible Ethernet packet. Additionally, the total buffer length must be a multiple of 8 bytes; otherwise NX/101 will reduce the buffer length to next lower multiple of eight.

In the reply message, the buffer length field total returns the number of bytes actually received, including 4 bytes for the CRC field. Note that if the buffer supplied was smaller than the packet received, then the excess bytes are truncated, and the buffer length will not give the true length of the packet.

5.3.8. Data Address Field

The data address field specifies the address of a buffer block in shared memory, where up to eight blocks compose a buffer. Note that the packet returned by the EXOS/101 does not include a preamble, so that the address of the first block will point to the first byte of the packet's destination field. The data address field is preserved in the reply message.

5.4. NET MODE Request/Reply Message

#	Length	Offset	Field Name	Request	Reply
1)	2	0	Reserved for NX Usage	zero	undefined
2)	4	2	User Id Code	undefined	preserved
3)	1	6	Request Code	08H	preserved
4)	1	7	Return Code (—,AL)	undefined	see text
5)	1	8	Request Mask (DL,—)	see text	undefined
6)	1	9	Options Mask (CL,CL)	see text	see text
7)	1	10	Mode (DH,DH)	see text	see text

|<-----1 byte----->|

Figure 5-5: NET MODE Request/Reply Message

The NET_MODE request is used to read/write the network controller mode and options mask objects. For details of these, see sections 7.3 and 7.4. Figure 5-5 shows the format of the NET_MODE request/reply message, and the following paragraphs describe its individual fields in detail.

5.4.1. Reserved Field

The first field is reserved for use by NX/101, and must be set to 0. Its value in the reply message is undefined.

5.4.2. User Id Code Field

The user id code field is not interpreted by the EXOS/101, and is returned unmodified in the reply message. It can be used to establish a correspondence between request and reply messages.

5.4.3. Request Code Field

The request code field defines the request. Its value in the NET_MODE request message must be 08H. This value is preserved in the reply message.

5.4.4. Return Code Field

The return code field is undefined in the request message. In the reply message, it reports the status of the NET_MODE request:

0 successful completion.

0A1H failed, the EXOS/101 is not in controller mode.

5.4.5. Request Mask Field

The request mask field is defined as follows:

01 write request bit.

02 read request bit.

Read and write can be requested simultaneously (mask = 03). Other bits in the mask must be 0, or effects are undefined.

The request mask's value in the reply message is undefined.

5.4.6. Options Mask Field

The options mask field defines several available controller options. Available options are defined by the following bit OR-able values:

10H alignment error - enables reception of packets even if the number of bits received is not a multiple of 8.

20H CRC error - enables reception of packets even if the CRC check fails.

80H net disable - disables the Ethernet controller so that packets are not received or transmitted on the Ethernet. However, transmit requests are still processed by NX/101, and to user processes appear to complete successfully if an on net mode is selected.

All other bits are undefined and must be 0. This parameter is required only if a write is requested. If the read bit in the request mask of the request message was set, then this field returns the options mask prior to the request. Otherwise its value in the reply message is undefined.

5.4.7. Mode Field

The mode field specifies the new mode of the Ethernet controller. Possible values are:

00H disconnect from the net.

01H connect to net, perfect filter for multicast addresses.

EXOS/101: Link Level Controller Mode

02H connect to net, only hardware filter for multicast addresses.

03H connect to net, receive all packets (promiscuous mode).

This parameter is required only if a write is requested. If the read bit in the request mask of the request message was set, then this field returns the net mode prior to the request. Otherwise its value in the reply message is undefined.

5.5. NET ADDRS Request/Reply Message

#	Length	Offset	Field Name	Request	Reply
1)	2	0	Reserved for NX Usage	zero	undefined
2)	4	2	User Id Code	undefined	preserved
3)	1	6	Request Code	09H	preserved
4)	1	7	Return Code (—,AL)	undefined	see text
5)	1	8	Request Mask (DL,DL)	see text	see text
6)	1	9	Address Slot (DH,—)	see text	preserved
7)	6	10	Net Address (*ES+DI,—)	see text	see text

|<-----1 byte----->|

Figure 5-6: NET ADDRS Request/Reply Message

The NET_ADDRS request is used to read/write an address in a specified address slot. For information about address slots, see section 7.5.

If a network address to be written is invalid, the write does not occur, and the address in the slot prior to the request is preserved. Writing an address into a slot disables reception on that slot. The NET_RCV request must be explicitly used to re-enable reception on the slot.

Figure 5-6 shows the format of the NET_ADDRS request/reply message, and the following paragraphs describe its individual fields in detail.

5.5.1. Reserved Field

The first field is reserved for use by NX/101, and must be set to 0. Its value in the reply message is undefined.

5.5.2. User Id Code Field

The user id code field is not interpreted by the EXOS/101, and is returned unmodified in the reply message. It can be used to establish a correspondence between request and reply messages.

5.5.3. Request Code Field

The request code field defines the request. Its value in the NET_ADDRS request message must be 09H. This value is preserved in the reply message.

5.5.4. Return Code Field

The return code field is undefined in the request message. In the reply message, it reports the status of the NET_ADDRS request:

0 successful completion.

0D1H the specified slot does not exist or access is not permitted.

0D3H improper address. Multicast slots can only take multicast addresses and the physical slot can only take a physical address. Attempting to write the broadcast slot (number 255) results in this error.

0A1H failed, the EXOS/101 is not in controller mode.

5.5.5. Request Mask Field

The request mask field is defined in the request message as follows:

01 write request bit.

02 read request bit.

Read and write can be requested simultaneously (mask = 03). Other bits in the mask must be 0, or effects are undefined.

In the reply message, if bit 3 (mask value 8) is set, then the address slot contained a valid address prior to this request. Otherwise the slot was empty. All other bits are undefined. This result is defined only if a read was requested.

5.5.6. Address Slot Field

The address slot field designates the address slot which is to be accessed. This can be the physical address slot (253) or any multicast address slot (between 1 and the limit defined by configuration).

This field's value is preserved in the reply message.

5.5.7. Net Address Field

The net address field, if a write is requested, should contain a valid network address to be written in the specified slot. In the reply message, if a read

EXOS/101: Link Level Controller Mode

was requested, and the slot was not empty, then this field returns the net address in the specified slot prior to this request. Otherwise it is undefined.

5.6. NET_RECV Request/Reply Message

#	Length	Offset	Field Name	Request	Reply
1)	2	0	Reserved for NX Usage	zero	undefined
2)	4	2	User Id Code	undefined	preserved
3)	1	6	Request Code	OAH	preserved
4)	1	7	Return Code (--,AL)	undefined	see text
5)	1	8	Request Mask (DL,DL)	see text	see text
6)	1	9	Address Slot (DH,--)	see text	preserved

|<-----1 byte----->|

Figure 5-7: NET_RECV Request/Reply Message

This request is used to read/alter the receive status of an address slot (see section 7.5).

Figure 5-7 shows the format of the NET_RECV request/reply message, and the following paragraphs describe its individual fields in detail.

5.6.1. Reserved Field

The first field is reserved for use by NX/101, and must be set to 0. Its value in the reply message is undefined.

5.6.2. User Id Code Field

The user id code field is not interpreted by the EXOS/101, and is returned unmodified in the reply message. It can be used to establish a correspondence between request and reply messages.

5.6.3. Request Code Field

The request code field defines the request. Its value in the NET_RECV request message must be OAH. This value is preserved in the reply message.

5.6.4. Return Code Field

The return code field is undefined in the request message. In the reply message, it reports the status of the NET_RECV request:

- 0 successful completion.
- 0D1H the specified slot does not exist or access is not permitted.
- 0D2H the specified slot was empty.
- 0A1H failed, the EXOS/101 is not in controller mode.

5.6.5. Request Mask Field

The request mask field is defined in the request message as follows:

- 01 write request bit.
- 02 read request bit.
- 04 enable receive bit.

Read and write can be requested simultaneously (mask = 03). Other bits in the mask must be 0, or effects are undefined.

If the write bit in the request mask is set, then reception on the designated address slot will be enabled or disabled, depending on the value of the enable receive bit.

In the reply message, if bit 2 (mask value 4) is set, then receive was enabled for this slot prior to this request. Otherwise it was disabled. All other bits are undefined. This result is defined only if a read was requested.

5.6.6. Address Slot Field

The address slot field designates the address slot which this request will work on. This can be the physical address slot (253), the broadcast slot (255), or any multicast address slot (between 1 and the limit defined by configuration).

This field's value is preserved in the reply message.

5.7. NET STSTCS Request/Reply Message

#	Length	Offset	Field Name	Request	Reply
1)	2	0	Reserved for NX Usage	zero	undefined
2)	4	2	User Id Code	undefined	preserved
3)	1	6	Request Code	OBH	preserved
4)	1	7	Return Code (—,AL)	undefined	see text
5)	1	8	Request Mask (DL,—)	see text	undefined
6)	1	9	Reserved	zero	undefined
7)	2	10	Number of Objects (CX,CX)	see text	see text
8)	2	12	Objects Index (SI,—)	see text	preserved
9)	4	14	Buffer Address (*ES+DI,—)	see text	preserved

|<-----1 byte----->|

Figure 5-8: NET STSTCS Request/Reply Message

This request reads/resets the statistics objects (see section 7.6). If the read bit is set in the request mask, then a specified number of statistics objects, starting at the objects index field, are copied into the array specified by the buffer address field. Note that the statistics copied into host memory are defined only after the reply message has been received.

If the write bit is set in the request mask, then the number of objects specified by the number of objects field, starting with the object specified by the objects index, are reset to zero. If the objects index field is out of range, then no objects are read/reset.

Figure 5-8 shows the format of the NET_STSTCS request/reply message, and the following paragraphs describe its individual fields in detail.

5.7.1. Reserved Field

The first field is reserved for use by NX/101, and must be set to 0. Its value in the reply message is undefined.

5.7.2. User Id Code Field

The user id code field is not interpreted by the EXOS/101, and is returned unmodified in the reply message. It can be used to establish a correspondence between request and reply messages.

5.7.3. Request Code Field

The request code field defines the request. Its value in the NET_STSTCS request message must be 0BH. This value is preserved in the reply message.

5.7.4. Return Code Field

The return code field is undefined in the request message. In the reply message, it reports the status of the NET_STSTCS request:

0 successful completion.

0A1H failed, the EXOS/101 is not in controller mode.

5.7.5. Request Mask Field

The request mask field is defined in the request message as follows:

01 write request bit.

02 read request bit.

Read and write can be requested simultaneously (mask = 03). Other bits in the mask must be 0, or effects are undefined.

The read request copies the specified portion of the statistics array into the specified buffer. The write request resets the specified portion of the statistics array. If both read and write are requested, the read is done first. This field is undefined in the reply message.

5.7.6. Reserved Field

This field must be zero in the request message. Its value in the reply message is undefined.

5.7.7. Number of Objects Field

The number of objects field specifies how many statistics objects are to be read/reset. In the reply message, this field returns the number of objects that were actually read/reset. If the number requested exceeds the bounds of the statistics array, it will be truncated.

5.7.8. Objects Index Field

The objects index field specifies the starting place in the statistics array at which objects will be read/reset. Its value is preserved in the reply message.

5.7.9. Buffer Address Field

The buffer address field specifies the address of the buffer in shared memory to which the requested portion of the statistics object array will be copied, if a read request was made. This field is defined only if a read is requested. Its value is preserved in the reply message.

6. THE NX/101 PROGRAMMING ENVIRONMENT

This section provides information necessary to write higher-level software to run under the NX/101 kernel on an EXOS/101 Ethernet front-end processor. The first few sections describe environmental considerations, such as memory allocation, which commonly affect software design. Subsequent sections explain the abstract objects and operations implemented in NX/101.

6.1. Overview

All programs for the Excelan Ethernet network processor board (EXOS/101) run on an Intel 8088 CPU under an EPROM-resident multi-tasking operating system kernel (NX/101). Programs can be written in any language for the 8088 and can be located anywhere in the memory available to the user. They can be downloaded either from the host or over the network. The procedure for downloading programs is described in section 4.8 of this manual.

NX/101's multi-tasking environment facilitates the structured implementation of high-level protocol software, as a set of cooperating processes. Facilities include mechanisms for process synchronization, interprocess communication, scheduling, and clock-based functions. None of the hardware devices on the board, viz., the clock, the Host interface or the Ethernet controller, require direct access by user processes. Instead, NX/101 has built-in drivers which provide suitable abstractions of the devices, so that programs developed for the EXOS/101 are independent of actual hardware implementation.

All functions of NX/101 are accessed by means of NX/101 calls executed by an INT n instruction, where n defines the desired function. Parameters to the calls are generally passed in CPU registers. However, it is easy to write interface libraries to permit NX calls to be made from programs written in high level languages such as C, PASCAL, etc.

6.2. Memory Organization

The 8088 provides an address space of 1 Mbyte, accessible in 64K segments, on 16-byte bounds. Figure 6-1 shows how this address space is allocated on the EXOS/101, under the default configuration of NX/101. The default configuration provides a given number of objects, such as mailboxes and process table entries. This allocation (specified in section 4.4) should be sufficient for most applications. However, the allocation of objects under NX/101 can be changed at initialization time, with a corresponding effect on RAM allocation. The following paragraphs explain the use of EXOS/101 memory in detail.

6.2.1. Interrupt Vector Table

In the default configuration, NX/101 allocates 512 bytes for the interrupt vector table, providing 128 entries of 4 bytes each. Of these, 32 interrupt vectors are available for user definition. If more are required, the interrupt vector table may be reconfigured to its full size of 1024 bytes. Interrupt allocation is explained below in more detail.

<u>Address</u>	<u>Function</u>	<u>Size</u>
FFFFFFH	Reserved Address Space	F0000H (960 Kbyte)
:	:	:
:	:	:
1FFFFFH	Single-Ported RAM (Model 2)	10000H (64 Kbyte)
:	or	:
:	Reserved (Model 1)	:
0FFFFFH	Dual-Ported User RAM	0F000H (60 Kbyte)
:	:	:
00FFFFH	Fixed NX/101 Data Area	00C00H (3 Kbyte)
003FFH	Movable NX/101 Data Area	00200H (1/2 Kbyte)
001FFH	Interrupt Vector Table	00200H (1/2 Kbyte)
00000		
	<-----1 byte----->	

Figure 6-1: Default EXOS/101 Memory Allocation

6.2.2. Movable NX/101 Data Area

The movable NX/101 data area is the memory required for data structures which NX/101 associates with objects. User software should not access this memory area directly.

The length of the movable data area depends on the maximum number of objects supported, which is configured during NX/101 initialization (see section 4). It can be computed by the expression $16*(P+M)+8*A$ bytes, where P is the number of processes, M is the number of the mailboxes and A is the number of address slots. In the default configuration this area is 512 bytes long, occupying locations 200H through 3FFH.

If NX/101 is reconfigured such that this area requires more than 512 bytes, or if locations 200H to 3FFH are needed for an expanded interrupt vector table, then this area can be moved to any memory area between 1000H and 0FFFFH.

6.2.3. Fixed NX/101 Data Area

NX/101 uses this memory area for data structures that are not dependent on its configuration. It is always 512 bytes long and occupies locations 400H through 0FFFH. It cannot be moved. User software should not directly access the fixed data area.

6.2.4. Dual-Ported User RAM

64 Kbytes of RAM on the EXOS/101, from location 0 to 0FFFFH, is dual-ported between the 8088 CPU and the Ethernet controller hardware. Of this, 60 Kbytes between 1000H and 0FFFFH, are entirely available in the default configuration for the purposes of down-loaded user software. If the movable data area must be moved from its default location, then some small portion of this RAM will become unavailable for user software. NX/101 requires that all message buffers (used for communicating data between processes, host, and network) lie in dual-ported user RAM.

6.2.5. Single-Ported RAM

On the EXOS/101 Model 2 only, another 64 Kbytes of RAM, from location 10000H to 1FFFFH is also available for the purposes of down-loaded software. This differs from the dual-ported user RAM only in that it may not be used for message buffers.

6.2.6. Reserved Address Space

The address range 10000H-FFFFFH on the Model 1, or 20000H-FFFFFH on the Model 2, is reserved. The effects of access to this area by user software are not defined.

Other than those described above, NX/101 imposes no restrictions on how memory is used. Users can link and load their programs in any manner they please. NX/101 does not define any buffer management services; users may choose the optimum scheme for individual applications.

6.3. Interrupt Types

The 8088 CPU provides 256 interrupt types, each of which corresponds to a 4-byte interrupt vector table entry.

NX/101 allocates these as follows:

- 0-31 reserved/dedicated by Intel.
- 32-63 device interrupts (used by NX/101).
- 64-95 NX/101 calls.
- 96-127 available to user software by default.
- 128-255 available to user software by reconfiguration.

User software should not modify interrupt vectors for types 0-95. In the default configuration, types 96-127 are available for user definition. If

more interrupt types are required, then the movable data area can be relocated (see section 4.3.11), making types 128-255 available.

NX/101 provides all interrupt service routines necessary for EXOS/101 hardware and the host interface. Therefore it is unlikely that user applications would require hardware interrupt service routines. However, it may be convenient to use the user-definable interrupt types as an interface between user software modules. If this is done, then the software interrupt service routines should be sure to re-enable interrupts immediately upon entry.

6.4. Processes

NX/101 supports processes as they are usually understood: a program in execution. Processes can be freely created and deleted. At any one time the number of processes cannot exceed a maximum number defined by the configuration of NX/101 (see section 4.4.12).

6.4.1. Process Address Space

NX/101 does not impose any memory protection between processes. All processes share the same 1-Mbyte address space, allowing them to communicate via shared memory. However, the context of each process includes the segment register file of the 8088. Thus each process can independently choose its own direct address space. The 8088 memory addressing architecture permits shared-text processes and dynamic relocation of code modules.

6.4.2. Process-id

Each process is identified by a unique one-word integer called its process-id. This number is used to refer to processes in all NX/101 calls. The process-id of a process which has been deleted is not re-used until at least 255 additional processes have been created after the deletion. Applications which create and delete processes very frequently should beware of this fact. The process-id 0 is a special id and always refers to the process currently running. Thus a process can refer to itself by using 0 instead of its actual id in an NX/101 call. When a process is first created its id is available in one of its CPU registers, as specified in the PROC_CREATE call description. A process can also find its own id by executing a read-only call (such as PROC_PRIOR), specifying 0 as the process-id parameter. All NX/101 calls always return the actual process-id even if 0 was used as an input parameter.

6.4.3. Process Stack

The stack address of a new process is supplied as a parameter to the PROC_CREATE call, by the process invoking this call. Note that NX/101 creates the first process, and allocates its stack within the fixed NX/101 data area (see section 6.2). Stack areas can be allocated anywhere in memory. When deciding stack size, the user should be aware that NX/101 does not maintain any separate system stack for a process. When NX/101 services interrupts, it uses the stack of the process running when the interrupt occurs. In order to prevent stack overflows it is recommended that user process stack size be such that at least 64 bytes of the stack is always available for NX/101 interrupt service routines.

6.4.4. Process Scheduling

Four parameters visible to user software drive NX/101's process scheduling algorithm:

priority
time slice
time count
sleep count

All but time count are explicitly set when a process is created, and can be examined or modified by any process subsequently. Time count can be examined, but its value is implicitly determined by time slice.

Priority is a number between 0 and 255 where 0 is the lowest priority. NX/101 maintains a logically separate scheduling queue of processes for each priority level. Process priority remains constant, unless modified by an explicit call to PROC_PRIOR.

Time slice is a number between 0 and 255 which determines the amount of CPU time that a process can use before its position in the scheduling queue is re-evaluated. Implementation is as follows. Time count is initialized with the value of time slice, and counts down at the rate of 20 milliseconds, but only while the process is actually running. When time count reaches 0, the process is put at the end of its scheduling queue, and time count is reinitialized with time slice. A process's time slice remains constant, unless modified by an explicit call to PROC_TIMSLC. A value of -1 for time slice is considered infinity.

Sleep count is a number between 0 and 64K-1 that represents the amount of real time which must elapse before the process becomes eligible to use the CPU. A process having a non-zero sleep count is said to be sleeping and a process with a sleep count of zero is said to be runnable. Sleep count also counts down at a rate of 20 milliseconds, and a value of -1 is considered infinity. By definition, however, the sleep count counts down only while the process is not running. When sleep count reaches zero, it remains zero and the process becomes runnable.

The scheduling parameters are used to implement a pre-emptive round robin scheduling algorithm as follows. Whenever scheduling occurs, the CPU is given to the first runnable process in the highest priority scheduling queue. Scheduling occurs on every tick of the NX/101 clock (20 milliseconds), and whenever some scheduling parameter changes. For instance, if during the execution of a process some other process with a higher priority becomes runnable, then the CPU is immediately given to the higher priority process without changing the position or time slice count of the pre-empted process. Similarly, if the sleep count of a running or a runnable process is set to a non-zero value either by an explicit NX/101 call or by an implicit side effect of some other NX/101 call, then the process is put to sleep without changing its time slice count or position in its priority queue.

It should be noted from the above discussion that a runnable process cannot have a non-zero sleep count. Thus setting the sleep count of a process to zero makes it runnable, and setting it non-zero suspends the process. The

PROC_SLPCNT call can be used by any process to alter the sleep count of any process. The new value overwrites the previous value of the sleep count, which is forgotten. By choosing an appropriate value for sleep count, a process can be delayed, suspended or resumed. As such, no separate calls for these capabilities are included in NX/101.

It should also be noted that if an infinite time slice is given to all processes then the scheduling policy reduces to a priority-based event driven scheduling algorithm. Running all processes at equal priority besides reduces the policy even further, to strictly event-driven scheduling.

6.4.5. Implicit Scheduling Factors

When a user process makes an NX/101 kernel call, it is locked until the call completes. Therefore the process will not be pre-empted by another user process unless the kernel call itself blocks the calling process. For instance, a MLBX_RECV call might cause re-scheduling before it completes, if no message is queued on the indicated mailbox. On the other hand, a MEM_READ or MEM_WRITE will always exclude other user processes until it completes.

NX/101 interrupt service routines will always interrupt any user process, and will interrupt each other according to this priority scheme:

- 0) Clock Tick
- 1) Ethernet Transmit Completion
- 2) Ethernet Receive Completion
- 3) Host Interface Event

where 0 is the highest priority. Note that any of these events can cause re-scheduling of user processes. For instance, an Ethernet Receive completion might place an Ethernet transmit reply message on a reply mailbox, and therefore reset the sleep count of a process enqueued there.

6.5. Mailboxes

Interprocess communication and synchronization is supported primarily by mailboxes. A mailbox, like a process, is an object that can be created and deleted. The maximum number of mailboxes that can exist at any given time is defined by the configuration of NX/101 (see section 4.4.13).

6.5.1. Mailbox-id

Each mailbox is identified by a unique one word integer called its mailbox-id. This number is used to refer to mailboxes in all NX/101 calls. When a mailbox is deleted its id is not re-used until at least 255 mailboxes have been created. Applications which create and delete mailboxes very frequently should beware of this fact.

6.5.2. Messages

A mailbox provides the facility to transfer messages between processes. A message is a memory-resident data structure with an arbitrary format, except for a mandatory 32-bit link field at its beginning. NX/101 uses the link field to chain messages. They must reside within the address range 0-0FFFFH

in EXOS/101 memory.

Since all processes share the same address space, messages are not copied by mailbox operations. Instead, pointers to messages are sent and received through the mailbox. It is the responsibility of the sending process to maintain the message data intact until the receiving process no longer requires it. The user must devise some scheme to ensure coordinated use of message data structures.

6.5.3. Null Messages

The null message is a special case which is identified by a null pointer and does not have any data associated with it. They are used strictly for process synchronization purposes, and can share a mailbox with regular messages. Note that null messages cannot be differentiated from one another.

6.5.4. Sending and Receiving Messages

The sending and receiving of messages through a mailbox is fully synchronized. Each mailbox has a message queue and a process queue. When a mailbox is created (using the `MLBX_CREATE` call) the process queue is empty and the message queue contains a specified number of null messages.

A message is sent to a mailbox by the `MLBX_SEND` call. When a regular message is sent it is appended after all other regular messages in the message queue but in front of all the null messages, if any. A null message is always appended after all the regular messages in the queue. Thus regular messages are delivered on a first-in first-out basis while null messages are delivered if and only if there are no regular messages in the queue.

A message is received from the mailbox by the `MLBX_RECV` call. A process receives the first message from the message queue if it is not empty. Otherwise, the process is appended to the end of the process queue and its sleep count is set to the value specified as a parameter to the `MLBX_RECV` call. Recall that if the sleep count of a process is nonzero the process is blocked until it counts down to zero. A subsequent `MLBX_SEND` call removes the first waiting process from the process queue, hands it the message and unblocks it by setting its sleep count to zero. When the unblocked process resumes execution, the `MLBX_RECV` call which blocked the process returns with a completion code indicating success.

If the sleep count of a blocked process counts down to zero or is explicitly set to zero by a `PROC_SLPcnt` call then the process is forced to unblock even if no message has arrived. In this case the unblocked process returns from the `MLBX_RECV` call with a nonzero completion code indicating the time out condition.

It should be noted from the above discussion that a process blocks on a mailbox only when the sleep count of the process is non-zero. Thus if a process executes a `MLBX_RECV` call with the sleep count parameter equal to zero, then the process will not block even if no messages are available. By choosing an appropriate value of the sleep count parameter, a process can test the availability of a message without blocking, block unconditionally until a message arrives, or block for a finite specified amount of time waiting for the

message to arrive.

6.5.5. Mailboxes as Semaphores

The notion of null messages allows the mailbox to be used as a conventional semaphore. If only null messages are used then the `MLBX_SEND` call is equivalent to the V operation and the `MLBX_RECV` call is equivalent to the P operation. Thus a mailbox can be used both for synchronizing a producer-consumer relationship or for mutual exclusion.

6.6. Process Locks

Using the mailbox for mutual exclusion may be a little more expensive than desired for simple cases such as updating a single variable. NX/101 provides the process lock as a simpler, alternate mechanism for mutual exclusion. A lock, when in effect, causes scheduling to be suspended. The call `PROC_LOCK` puts a lock in effect and the call `PROC_UNLOCK` removes a lock. Lock calls can be nested up to 32K deep; before a process is unlocked, unlock calls must balance lock calls. If a process with locks in effect makes an NX/101 call which can potentially cause the process to sleep then all locks are removed regardless of whether the process actually went to sleep or not.

The process executing a lock call excludes all other process from running and thus imposes a stronger condition than the mailbox mechanism, which excludes only the processes that intend to use the critical section. On the other hand, the lock call executes faster than the mailbox calls, and a lock does not consume memory resource as does a mailbox object. The lock call cannot be used for a producer-consumer type of synchronization. For mutual exclusion the users can select the mechanism which best suits the application.

Both mailboxes and locks provide mutual exclusion between processes; however, interrupts are not excluded. As such the only way to share a critical section between a process and an interrupt service routine is to disable interrupts for the duration of the critical section. Programmers usually need not be concerned with this fact, since all necessary interrupt handlers are included in NX/101. In general the user programs should not disable interrupts.

6.7. System Mailboxes

Certain NX/101 services are asynchronous by nature, such as sending and receiving messages with Ethernet or the host. All such system services are provided in a conceptual sense by system processes. These are like user processes in that they execute asynchronously, but they have no process-id or visible scheduling parameters. User processes access system process services by sending a request message to a special "system mailbox" associated with the desired service.

System mailboxes are created by NX/101 during initialization and are not included in the number of user mailboxes specified by the configuration of NX/101. The set of mailbox-ids for regular mailboxes is disjoint with the set of mailbox-ids for system mailboxes. NX/101 designates the following system mailbox-ids:

0001H Host
0009H Ethernet

Request messages are sent and received through a system mailbox with the regular `MLBX_SEND` and `MLBX_RECV` calls. After sending a request message, a process can continue to run or wait until the message is returned. The request message, once sent, should not be modified until the reply message is received.

The formats of request messages and their corresponding reply messages are defined by each individual service. All messages, however, have a standard header. Figure 6-2 shows this header, and the following paragraphs explain each field in detail.

#	Length	Offset	Field Name	Request	Reply
1)	4	0	Link	undefined	undefined
2)	2	4	Reply Mailbox	see text	see text
3)	1	6	Request Code	see text	see text
4)	1	7	Return Code	see text	see text
			: Additional Fields Defined by : Individual System Processes : :		
			<-----1 byte----->		

Figure 6-2: Standard Header for System Messages

6.7.1. Link Field

The link field is required by NX/101 at the beginning of all messages. Its request and reply values are both undefined. NX/101 uses this field for chaining messages.

6.7.2. Reply Mailbox Field

The reply mailbox field specifies the mailbox to which the request message is returned after the completion of the requested service. System mailboxes cannot be used as reply mailboxes.

6.7.3. Request Code Field

The request code field specifies the service to be performed, typically read or write.

6.7.4. Return Code Field

The return code field is the result of the request filled in by the system process. The actual values for the request and return codes are defined by individual system services.

6.8. The Clock Device

NX/101's abstraction of the clock device is a simple 64-bit counter, incremented in real time every 20 milliseconds. On reset the counter is set to zero. Processes can set the clock counter to any value at any time with the `TIME_SET` call, and read it at any time with the `TIME_GET` call. The clock counter can be used for any purpose required by the user application. For example, it may be used as a time-of-day clock by setting it to the current time.

Note this model of the clock provides no facility to the user for interrupting after a specified interval of time. This clock-related function has been incorporated directly in NX/101's multi-tasking model by the sleep count parameter, which can be used to delay a process or force a blocked process to unblock after a specified interval of time.

7. THE NX/101 ETHERNET INTERFACE

The NX/101 Ethernet interface consists of two parts: a system process which sends and receives packets, and several NX/101 kernel calls which serve network management functions. This section describes all necessary details of the Ethernet system process, and describes the functionality of the network management calls. For further details about NX/101 call format, see section 9.

User processes send Ethernet transmit and receive requests to the Ethernet system process via the Ethernet system mailbox (0009H). The transmit request describes the location of a packet to transmit, while the receive request describes a buffer in which to store an incoming packet. In both cases, the request names a reply mailbox, to which the system process sends a reply message when the request completes. Requests may be enqueued without limit, and will be processed asynchronously. Transmit requests are serviced as rapidly as the Data Link protocol permits, and return a failure only in the event of excess collisions. Receive requests return a reply message only when an incoming packet satisfies all specified address filtering.

Network management functions determine the Ethernet controller's mode, define which addresses to accept, and gather network statistics. All of these are defined in terms of abstract objects, accessed only via the appropriate NX/101 calls. In each call, a request mask parameter determines whether the request will read or write (or read and write) a value to/from a given object. This approach simplifies access to network management function, and insulates the functions from specific implementation methods.

7.1. Ethernet Transmit Request

In order to send a packet on the Ethernet, a process sends a service request message to the Ethernet system mailbox. When transmission is complete, the request message (modified according to the status of the transmission) is returned to a reply mailbox designated by the requesting process. The request message does not contain the actual data to be sent, but rather a pointer to the packet. Any number of messages can be sent to the Ethernet system process; they will be queued up and dispatched in the order received. Until the reply message is received, the message and packet belong to the Ethernet process, and should not be modified.

Transmit requests are serviced immediately, unless the controller is in off net mode (see section 7.3). When a NET_MODE call places the controller off net, any transmission underway will complete, but any enqueued requests will remain enqueued. When off net, new transmit requests may still be enqueued. When the controller is restored to an on net mode, transmission resumes.

If the net disable option is selected, (see section 7.4) then transmission will appear to proceed normally, but nothing is actually transmitted on the Ethernet.

Packets are prepared for transmission in standard Ethernet data link layer frame format, as shown in figure 7-1. However, the packet need not include a frame check sequence (CRC) field. This, and the preamble, are generated by EXOS/101 hardware.

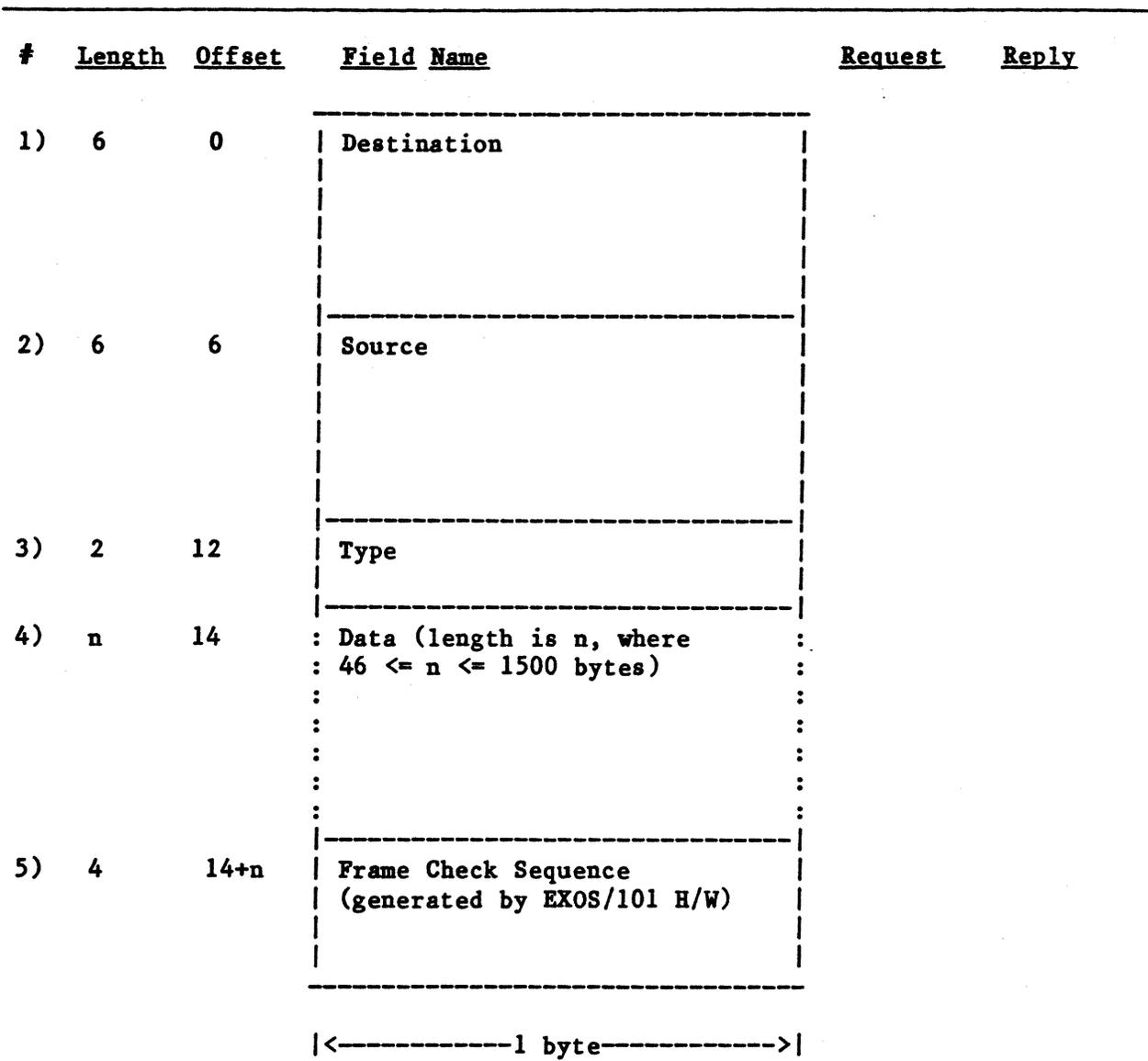


Figure 7-1: Ethernet Packet Format

Figure 7-2 shows the format of an Ethernet transmit request/reply message. Its fields are explained in detail below.

7.1.1. Link Field

The link field is required by NX/101 at the beginning of all messages. Its request and reply values are both undefined. NX/101 uses this field for chaining messages.

7.1.2. Reply Mailbox Field

The reply mailbox field identifies the mailbox to which the request message is returned after completion of the request. In the request message, this must

#	Length	Offset	Field Name	Request	Reply
1)	4	0	Link	undefined	undefined
2)	2	4	Reply Mailbox	see text	see text
3)	1	6	Request Code	zero	preserved
4)	1	7	Return Code	undefined	see text
5)	1	8	Address Slot	undefined	see text
6)	1	9	Reserved	zero	undefined
7)	2	10	Data Length	see text	undefined
8)	4	12	Data Address	see text	preserved

|<-----1 byte----->|

Figure 7-2: Ethernet Transmit Request/Reply Message

identify an existing user mailbox. Its value in the reply message is the Ethernet system mailbox id.

7.1.3. Request Code Field

The request code field defines the request; in this case, to send a packet, its value in the request message must be 0. The reply message preserves this value.

7.1.4. Return Code Field

The return code field value is undefined in the request message. In the reply message, it reports the status of the transmission request:

00H successful transmission, no retry.

01H successful transmission, 1 retry.

02H successful transmission, more than 1 retry.

10H transmission failed, excessive collisions.

40H transmission failed, transmit length not in range.

7.1.5. Address Slot Field

The address slot field is an index into the address slot array (see section 7.5). Its value in the request message is undefined. In the reply message, it contains the address slot number by which this packet would be received by this station. For instance, the value 255 indicates that the packet was broadcasted, and should be auto-received. Or, if the packet was transmitted to this station's own address, the value would be 253. A zero value means that no slot matched - this packet would not be received by this station.

7.1.6. Reserved Field

This field is reserved for future use. In the request message, its value must be 0. In the reply message, its value is undefined.

7.1.7. Data Length Field

The data length field is the length, in bytes, of the packet to be transmitted. This value does not include the preamble or CRC fields, which are appended by EXOS/101 hardware. In the reply message, the data length field's value is undefined.

7.1.8. Data Address Field

The data address field is the address of the packet to be transmitted. This is an segmented address (see section 3.9); the first word is an offset, the second word a segment base address. The EXOS/101 requires that the packet lie entirely within the address range 1000H-0FFFFH. Additionally, the segment base address must be 0. Note that the packet, as handed over to the Ethernet process, does not include a preamble, so that the address will point to the first byte of the packet's destination field. The data address field's value is preserved in the reply message.

7.2. Ethernet Receive Request

In order to receive a packet on the Ethernet, a process sends a service request message to the Ethernet System mailbox. The request message does not contain the actual buffer to be filled, but rather a pointer to the buffer. When reception is complete, the request message (modified according to the status of the reception) is returned to a reply mailbox designated by the requesting process. Once the reply message is received, the buffer belongs to the receiving process. Receive requests are not necessarily dispatched in the order they are received by the Ethernet system process.

The EXOS/101 manages receive buffer descriptors in hardware; it can receive packets back-to-back with minimum interframe spacing as long as sufficient receive requests have been enqueued. For most applications, it is recommended that at least two receive buffers be made available at all times. This allows

time for the EXOS CPU to screen out undesired packets (such as spurious network bootstrap protocol messages, or multicast packets which passed the hardware filter) which would otherwise tie up a single-buffered implementation. By queuing up a fairly large number of receive buffers, protocols can create a large "receive window" and realize substantial performance improvements.

Receive requests return a reply message to a designated reply mailbox only after an incoming packet satisfies the receive address filtering criteria. When a NET_MODE call (see section 9) places the controller off net, any receive underway will complete, but any enqueued requests will remain enqueued. Packets arriving on the Ethernet will be ignored (not placed into receive buffers). When off net, new receive requests may still be enqueued.

If the net disable option is selected, (see section 7.4) then incoming traffic is ignored, and receive requests will not return a reply message until the controller is re-enabled.

If the EXOS/101 was initialized in network bootstrap mode, once the network bootstrap session is completed, it will not pass messages of the network bootstrap type to software running under NX/101. This prevents any spurious network bootstrap messages from interfering with successfully-installed protocol software on the EXOS/101.

Received packets are in standard Ethernet data link layer frame format, as shown in figure 7-1. The frame check sequence (CRC) field is included.

Figure 7-3 shows the format of an Ethernet receive request/reply message, which is very much like the transmit request/reply message. Its fields are explained in detail below.

7.2.1. Link Field

The link field is required by NX/101 at the beginning of all messages. Its request and reply values are both undefined. NX/101 uses this field for chaining messages.

7.2.2. Reply Mailbox Field

The reply mailbox field identifies the mailbox to which the request message is returned after completion of the request. In the request message, this must identify an existing user mailbox. Its value in the reply message is the Ethernet system mailbox id.

7.2.3. Request Code Field

The request code field defines the request; in this case, to receive a packet, its value in the request message must be 1. The reply message preserves this value.

7.2.4. Return Code Field

The return code field value is undefined in the request message. In the reply message, it reports the status of the receive request:

#	Length	Offset	Field Name	Request	Reply
1)	4	0	Link	undefined	undefined
2)	2	4	Reply Mailbox	see text	see text
3)	1	6	Request Code	1	preserved
4)	1	7	Return Code	undefined	see text
5)	1	8	Address Slot	undefined	see text
6)	1	9	Reserved	zero	undefined
7)	2	10	Buffer Length	see text	see text
8)	4	12	Buffer Address	see text	preserved

|<-----1 byte----->|

Figure 7-3: Ethernet Receive Request/Reply Message

- 00H packet received with no error.
- 04H packet received longer than buffer supplied, truncated.
- 10H packet received with alignment error.
- 20H packet received with CRC error.
- 40H no packet received, buffer supplied was less than 64 bytes.

Note that packets with errors are actually received only if the network mode is set appropriately.

7.2.5. Address Slot Field

The address slot field is an index into the address slot array (see section 7.5). Its value in the request message is undefined. In the reply message, it contains the address slot number which matched the destination address of the packet received. If the controller is in promiscuous mode, then this

field will return the universal address slot, whether or not any address matched. If the controller is not in perfect filtering mode, then this field will return the universal address slot when any multicast packet is received.

7.2.6. Reserved Field

This field is reserved for future use. In the request message, its value must be 0. In the reply message, its value is undefined.

7.2.7. Buffer Length Field

The buffer length field is the length, in bytes, of the receive buffer. The length does not include the preamble but must include 4 bytes for the frame check sequence (CRC) field. In order to receive the longest possible Ethernet packet, the buffer must be at least 1520 bytes long. Minimum size is 64 bytes, which will fit the shortest possible Ethernet packet. Additionally, the buffer length must be a multiple of 8 bytes; otherwise NX/101 will reduce the buffer length to next lower multiple of eight.

In the reply message, the buffer length field returns the number of bytes actually received, including 4 bytes for the CRC field. Note that if the buffer supplied was smaller than the packet received, then the excess bytes are truncated, and the buffer length will not give the true length of the packet.

7.2.8. Buffer Address Field

The data address field is the buffer to receive a packet. This is an segmented address (see section 3.9); the first word is an offset, the second word a segment base address. The EXOS/101 requires that the buffer lie entirely within the address range 1000H-0FFFFH. Additionally, the segment base address must be 0. Note that the packet returned by the Ethernet process does not include a preamble field, so that the address will point to the first byte of the buffer's destination field. The buffer address field's value is preserved in the reply message.

7.3. Ethernet Controller Modes

The Ethernet controller provides several optional modes of operation which essentially define filtering criteria for packets to be received. Processes can read and modify the controller's mode with the NET_MODE call, which selects one of four mutually exclusive modes:

- 0 off net - the EXOS/101 is disconnected from the net. No transmission or reception of packets takes place. Note, however, that any transmission or reception in progress is completed before the network is actually disconnected. Transmit and receive requests can still be enqueued, and network management functions will be serviced.
- 1 on net, perfect filtering - the EXOS/101 is connected to the Ethernet. Multicast packets are received if and only if their destination addresses match one of the specified multicast addresses. A two level filter is employed - the first level in hardware, which

rejects a large fraction of the unwanted messages, and the second level in software, which traps the balance.

- 2 on net, imperfect filtering - the EXOS/101 is connected to the Ethernet. Only the hardware filter is used to select the desired multicast packets. It is possible to receive spurious multicast packets, which fall through the hardware filter.
- 3 on net, promiscuous mode - the EXOS/101 is connected to the Ethernet. All packets are received regardless of their destination address.

When the EXOS/101 is reset the controller comes up in mode 0 - disconnected. The user has to explicitly set the desired mode, with the NET_MODE call.

7.4. Ethernet Controller Option Mask

This object defines various options, useful for testing and diagnostic purposes. Available options are defined by the following bit OR-able values:

- 10H alignment error - enables reception of packets even if the number of bits received is not a multiple of 8.
- 20H CRC error - enables reception of packets even if the CRC check fails.
- 80H net disable - disables the Ethernet controller so that packets are not received or transmitted on the Ethernet. However, transmit requests are still processed by NX/101, and to user processes appear to complete successfully if an on net mode is selected.

When reception of flawed packets is enabled, the actual error in a bad packet can be determined from the return code field in the receive reply message.

7.5. Address Slots

Address slots identify the destination addresses for which packets on the Ethernet should be received by this station. Each slot holds a single six-byte Ethernet address. Reception can be enabled or disabled individually for each slot.

Slots are identified by a unique number between 0 and 255. They are designated for certain purposes, according to this number, as follows:

- 0 the null slot
- 1-252 multicast address slots
- 253 physical address slot
- 254 universal address slot

255 broadcast address slot

The actual number of multicast address slots is configurable. The default number, and the details of configuring NX/101, are given in section 4.4.14.

Every EXOS/101 is permanently assigned a physical address from within a contiguous block of Ethernet physical addresses allocated to Excelan. This address is unique over all Ethernets. When the EXOS/101 is reset, the physical address is copied from EPROM into the physical address slot, where it can be read and modified.

Processes can read and write most address slots with the NET_ADDR call, and enable or disable reception for any slot with the NET_RECV call. Only valid multicast addresses may be written in multicast address slots. Only a valid physical address may be written in the physical address slot. The address in the broadcast slot cannot be changed. Note that writing an address in a slot disables reception on the slot - an explicit NET_RECV call must then be made in order to enable reception. Enabling or disabling reception on an address slot does not affect the address contained in it.

When the EXOS/101 is initialized, all multicast address slots are empty. The physical address slot (number 253) contains the station's unique Ethernet address. The broadcast slot (number 255) contains the broadcast address. On initialization, reception is enabled on the physical and the broadcast slots.

Address slot numbers are used as short identifiers for the network addresses contained in the corresponding slots. For instance, when a packet is received, the receive reply message returns the address slot number whose address matched the destination address of the packet. Thus, a slot 253 would indicate that the packet arrived for the physical address - and a slot 255 would indicate a broadcast packet. The universal slot 254 is returned if the network is in promiscuous mode, or if a multicast message is received in imperfect filtering mode.

7.6. Net Statistics

The EXOS/101 supports network management functions by gathering statistics on network operations. The statistics appear to the user as an array of two-word (32-bit) objects. Each object represents a counter that keeps a tally of events or some other value of interest, as described below. When an event counter reaches its maximum value, further counting on the object is inhibited until it is reset.

Not all 32 bits of an object are necessarily used. The number of bits used by each object is included in the description of the objects below. The used bits are always the least significant bits of the object. The bits unused by an object are undefined, and may take any value.

Processes can read and reset objects with the NET_STSTCS call. An object is referred to by its index in the array, where the index to the first object is 0. Multiple objects may be accessed in a single call, if they are contiguous. Resetting an object changes its value to 0. Resetting the EXOS/101 resets all statistics objects - otherwise they are continuously maintained.

Included in the statistics array is a Time Domain Reflectometer. The TDR function aids system designers and installers locate faults in the Ethernet cable. Shorts and opens in the cable manifest themselves in reflections, causing persistent spurious collisions. The TDR object contains the time delay between the start of transmission and the detection of a collision, in units of 100 nanoseconds, for the last transmission that encountered a collision.

Statistics objects are listed below by index number, with a complete description.

- 0 Frames Sent No Errors - a 32-bit counter that counts the number of frames successfully transmitted with or without retries.
- 1 Frames Aborted Excess Collisions - a 32-bit counter that counts the number of transmissions that were aborted because 16 collisions were encountered.
- 2 Undefined - reserved for future use.
- 3 Time Domain Reflectometer - a 16-bit object that remembers the delay between the start of the last transmission that encountered an excess collision and the detection of that collision. The delay is counted in units of 100 nanoseconds.
- 4 Frames Received No Errors - a 32-bit counter that counts the number of error-free frames received.
- 5 Frames Received Alignment Error - a 32-bit counter that counts the number of frames received with an alignment error i.e. frames that are not an exact multiple of an 8 bits in length. This statistic is maintained whether or not reception with alignment errors is enabled in the options mask (see section 7.4).
- 6 Frames Received CRC Error - a 32-bit counter that counts the number of frames received which had CRC errors. This statistic is maintained whether or not reception with CRC errors is enabled in the options mask (see section 7.4).
- 7 Frames lost - a 32-bit counter that counts the number of frames which would normally have been received but were lost because no receive buffers were available.

8. THE NX/101 HOST INTERFACE

User software on the EXOS/101 communicates with the host through a system process in the NX/101 kernel, which transmits and receives messages to and from the host processor. Access to this process is through a system mailbox associated with the host interface (0001H). NX/101 also provides the MEM_READ and MEM_WRITE calls which access shared Multibus memory directly. This section describes these facilities as seen by a process on the EXOS/101. For information about initializing and using the message interface from the host processor, see section 4.5.

Messages are commonly used to synchronize a producer-consumer relationship with the host, and to exchange information with objects in host memory which are unknown to processes on the EXOS/101. Typically, messages contain control information and pointers to data buffers in host memory, which can then be directly transferred. This approach allows user processes running on the EXOS/101 to assemble a data packet from scattered locations in host memory - which saves the host having to copy scattered blocks into one contiguous buffer for transfer in a message.

8.1. Host Transmit Request

In order to transfer a message to the host, a process sends a service request message to the system mailbox associated with the host. When the transfer is complete, the request message (modified according to the status of the transfer) is returned to a reply mailbox designated by the requesting process. Any number of messages can be sent to the host interface system process; they will be queued up and dispatched in the order received. Until the reply message is received, the message belongs to the system process and should not be modified.

Figure 8-1 shows the format of a host transmit request/reply message. Its fields are explained in detail below.

8.1.1. Link Field

The link field is required by NX/101 at the beginning of all messages. Its request and reply values are both undefined. NX/101 uses this field for chaining messages.

8.1.2. Reply Mailbox Field

The reply mailbox field identifies the mailbox to which the request message is returned after completion of the request. In the request message, this must identify an existing user mailbox. Its value in the reply message is the host interface system mailbox id.

8.1.3. Request Code Field

The request code field defines the request; in this case, to transmit a message, its value in the request message must be 0. The reply message preserves this value.

#	Length	Offset	Field Name	Request	Reply
1)	4	0	Link	undefined	undefined
2)	2	4	Reply Mailbox	see text	see text
3)	1	6	Request Code	zero	preserved
4)	1	7	Return Code	undefined	see text
5)	2	8	Data Length	see text	see text
6)	n	10	: Data :	: see text :	: preserved :

|<-----1 byte----->|

Figure 8-1: Host Transmit Request/Reply Message

8.1.4. Return Code Field

The return code field value is undefined in the request message. In the reply message, it reports the status of the transmission request:

00H Successful transfer.

04H Transfer failed, host's receive buffer was shorter than the transmit length. Should this occur, the host still receives the message, but it is truncated.

8.1.5. Data Length Field

The data length field is the length, in bytes, of the data field to be transferred. Zero is a valid value. In the reply message, this field returns the number of bytes actually transferred.

8.1.6. Data Field

The data field is the actual message to be sent in the transmit request. The data can be any number of bytes as long as it lies entirely within the address range 0-0FFFFH. The format of this data is defined entirely by the user. If the host data order conversion option is selected, NX/101 will apply any conversion needed for the byte string data type. The data field's contents are preserved in the reply message.

8.2. Host Receive Request

In order to receive a message from the host, a process sends a service request message to the system mailbox associated with the host interface. When reception is complete, the request message (modified according to the status of the reception) is returned to a reply mailbox designated by the requesting process. Receive requests are queued up and dispatched in the order they are received by the host interface system process. Once the reply message is received, the buffer belongs to the receiving process.

Figure 8-2 shows the format of an host receive request/reply message, which is very much like the transmit request/reply message. Its fields are explained in detail below.

#	<u>Length</u>	<u>Offset</u>	<u>Field Name</u>	<u>Request</u>	<u>Reply</u>
1)	4	0	Link	undefined	undefined
2)	2	4	Reply Mailbox	see text	see text
3)	1	6	Request Code	1	preserved
4)	1	7	Return Code	undefined	see text
5)	2	8	Data Length	see text	see text
6)	n	10	: Data	undefined	see text
			:		
			<-----1 byte----->		

Figure 8-2: Host Receive Request/Reply Message

8.2.1. Link Field

The link field is required by NX/101 at the beginning of all messages. Its request and reply values are both undefined. NX/101 uses this field for chaining messages.

8.2.2. Reply Mailbox Field

The reply mailbox field identifies the mailbox to which the request message is returned after completion of the request. In the request message, this must identify an existing user mailbox. Its value in the reply message is the host

interface system mailbox id.

8.2.3. Request Code Field

The request code field defines the request; in this case, to receive a message, its value in the request message must be 1. The reply message preserves this value.

8.2.4. Return Code Field

The return code field value is undefined in the request message. In the reply message, it reports the status of the transmission request:

00H Successful transfer.

04H Transfer failed, receive buffer was shorter than the buffer sent by the host. Should this occur, the EXOS/101 still receives the message, but it is truncated.

8.2.5. Data Length Field

The data length field is the length, in bytes, of the buffer supplied in this message. Zero is a valid value. In the reply message, this field returns the number of bytes actually transferred. Zero is a possible value.

8.2.6. Data Field

The data field is the buffer into which data from the host will be copied. It can be any number of bytes as long as it lies entirely within address range 0-0FFFFH. The format of this data is defined by the user. If the host data order conversion option is selected, NX/101 will apply any conversion needed for the byte string data type.

8.3. Direct Access to Host System Memory

The EXOS/101 accesses Multibus memory by mapping part of its own CPU's address space into Multibus memory addresses. This is the underlying mechanism which NX/101 uses to implement the message transfer functions described above. User software can directly utilize this direct memory access mechanism without sacrificing portability by using NX/101's MEM_READ and MEM_WRITE calls.

These calls take an address in EXOS/101 memory, an address in host memory, and a data transfer length. NX/101 performs the appropriate mapping and executes the transfer. If the host data order conversion option is enabled, NX/101 will apply any conversion needed for the byte string data type.

8.4. Host Data Order Conversion

For the convenience of protocol software running on the EXOS/101, NX/101 provides calls which convert data between the host system's ordering and the 8088 CPU's native ordering. These calls, CVT_WORD and CVT_LWORD, work in conjunction with the host data order conversion option (see section 4.2). By incorporating the calls in EXOS-resident software, the user's software can be made independent of data ordering, both on the host system, and on the EXOS/101.

EXOS/101: The NX/101 Host Interface

When the host data conversion option is enabled, NX/101 sets up the CVT_WORD and CVT_LWORD calls according to the test pattern which the host system presents in the configuration message. User software running on the EXOS/101 can then use these calls to convert word and longword data objects passed through the data field in the standard host message queue, or via the MEM_READ and MEM_WRITE calls. Note that byte string conversion, if required, is done implicitly by the primitive transfer operations. CVT_WORD and CVT_LWORD do not repeat that conversion.

9. NX/101 KERNEL CALL REFERENCE

This section defines the specific format and usage of NX/101 kernel calls. User software running on the EXOS/101 should access all NX/101 services through these requests. For more information about the function of NX/101 kernel calls, see sections 6, 7, and 8.

Processes request NX/101 services through an INT n instruction, where n is the type of the desired call. Parameters are generally passed in registers, although some parameters can be pointers to other parameters in memory. Passing parameters in registers facilitates writing interfaces for different high level languages, which may have different calling conventions.

Most calls return a completion code in the register AL. A negative completion code implies an error, and a zero or positive value implies success. Unless otherwise stated, only the registers used for passing parameters and results are modified.

The following list summarizes the NX/101 calls, which are grouped according to the abstract objects on which they operate.

PROC_CREATE	create a process.
PROC_DELETE	delete a process.
PROC_SLPCNT	read/write sleep count of a process.
PROC_PRIOR	read/write priority of a process.
PROC_TIMSLC	read/write time slice of a process.
PROC_STATUS	read status of a process.
PROC_LOCK	lock a process.
PROC_UNLOCK	unlock a process.
MLBX_CREATE	create a mailbox.
MLBX_DELETE	delete a mailbox.
MLBX_SEND	send a message to a mailbox.
MLBX_RECV	receive a message from a mailbox.
TIME_GET	get the time.
TIME_SET	set the time.
NET_MODE	read/write the net mode.
NET_ADDR	read/write the net address in a slot.
NET_RECV	enable/disable receive for a slot.
NET_STSTCS	read/clear network statistics.
MEM_READ	read system memory.
MEM_WRITE	write system memory
CVT_WORD	convert data order of word operand.
CVT_LWORD	convert data order of longword operand.
VERSION	return EXOS/101 version number.

The remainder of this section describes the NX/101 calls individually, in the order given above. A standard format is used, as follows:

CALL NAME

INTERRUPT TYPE

Parameters:

specification of the call parameters.

Results:

specification of the call results.

Description:

specification of the call's purpose and effects.

PROC_CREATE

INT 64

Parameters:

- BX: the offset part of the starting address of the new process.
- ES: the segment part of the starting address of the new process.
- CX: the initial sleep count for the new process. A value of -1 (OFFFH) is considered infinity.
- DL: priority of the new process (0 is the lowest, 255 is the highest).
- DH: time slice of the new process in ticks of 20 milliseconds. A value of -1 (OFFH) is equivalent to an infinite time slice.
- SI: the offset part of the address of the top of the stack for the new process.
- DI: the segment part of the address of the top of the stack for the new process.

Results:

- AL: completion code:
 - 0 successful.
 - 0FOH failed, maximum number of processes allowed already exists.
- AH: undefined.
- BX: process-id of the new process, valid only if call is successful.

Description:

This call creates a new process with the specified parameters and returns its process-id. Note that the stack area can be allocated anywhere in user memory. The stack pointer specified should point to the top of the new process stack. The initial CPU register values for the new process are defined as follows:

AX: undefined.
BX: process-id of the process.
CX: undefined.
DX: undefined.
SP: offset for process top-of-stack (parameter SI).
BP: undefined.
SI: undefined.
DI: undefined.
CS: segment base for process code (parameter ES).
DS: undefined.
SS: segment base for process stack (parameter DI).
ES: undefined.
IP: offset for starting address (parameter BX).
FLAGS: interrupts enabled, rest undefined.

The successful completion of this call invokes an immediate scheduling decision. Thus if a process spawns another process with a zero initial sleep count and a higher priority than its own, control will be passed immediately to the new process at its starting address, before the calling process returns from the call.

PROC_DELETE

INT 65

Parameters:

BX: process-id (0 implies calling process).

Results:

AL: completion code:

0 successful deletion.

0F1H specified process does not exist.

AH: undefined.

Description:

The specified process is deleted if it exists. It is the responsibility of the programmer to ensure that no harmful effects of deleting the process will occur (e.g. a process owning a critical section should not be deleted etc.). If the process being deleted was waiting in a mailbox it is first removed from the mailbox's process queue. If a process has invoked locks, and deletes itself, then any locks are removed.

PROC_SLPCNT

INT 66

Parameters:

BX: process-id (0 implies the calling process).

CX: new sleep count for the process, in ticks of 20 milliseconds. The value -1 (OFFFFH) represents infinity. This parameter is required only if a write is requested.

DL: request mask:

01 write request bit.

02 read request bit.

Read and write can be requested simultaneously (DL = 03). Other bits in mask must be 0, or effects are undefined.

Results:

AL: completion code:

0 successful completion.

0F1H failed, specified process does not exist.

AH: undefined.

BX: process-id of the specified process, not destroyed if the call fails.

CX: sleep count of the process just prior to this call. This result is defined only if the request mask (DL) had the read bit set and the call was successful.

Description:

This call is used to read/write the sleep count of the specified process. If the write bit in the request mask (DL) is set then the current value of the sleep count is replaced by the specified new sleep count (CX). If the read bit in the request mask (DL) is set, then the the value of the sleep count prior to this call is returned.

If modified, the new value of sleep count is put into effect immediately and thus may invoke rescheduling. If the sleep count of a blocked process is changed to 0 then it is unblocked even if the process was waiting for a message to arrive at some mailbox. This call can be used to delay, suspend or resume a process by setting the sleep count to non-zero, infinity, or zero respectively.

If this call changes the sleep count of the running process to non-zero then any locks in effect are canceled, regardless of errors.

PROC_PRIOR

INT 67

Parameters:

BX: process-id (0 implies the calling process).

DH: new priority of the process (required only if write is requested),
The lowest priority is 0 and the highest priority is 255.

DL: request mask:

01 write request bit.

02 read request bit.

Read and write can be requested simultaneously (DL = 03). Other bits in mask must be 0, or effects are undefined.

Results:

AL: completion code:

0 successful completion.

0F1H failed, specified process does not exist.

AH: undefined.

BX: process-id of the specified process, if the call succeeds. Otherwise its value before the call is preserved.

DH: priority the process prior to this call. This result is defined only if the request mask (DL) had the read bit set and the call was successful.

Description:

This call is used to read/write the priority of the specified process. If the write bit in the request mask (DL) is set, then the current priority of the process is replaced by the new specified priority (DH). If the read bit in the request mask is set, then the priority of the process prior to this call is returned. If modified, the new value of priority is put into effect immediately and re-scheduling is invoked. Thus if a process is lowering its own priority and a process with equal or higher priority is runnable, the call may not immediately return.

PROC_TIMSLC

INT 68

Parameters:

BX: process-id (0 implies the calling process).

DL: request mask:

01 write request bit.

02 read request bit.

Read and write can be requested simultaneously (DL = 03). Other bits in mask must be 0, or effects are undefined.

DH: new time slice of the process (required only if write is requested) in ticks of 20 milliseconds. A value of -1 (OFFH) represents infinity.

Results:

AL: completion code:

0 successful completion.

0F1H failed, specified process does not exist.

AH: undefined.

BX: process-id of the specified process if the call succeeds, otherwise not destroyed.

DL: time count the process prior to this call. This result is defined only if the request mask (DL) had the read bit set and the call was successful.

DH: time slice the process prior to this call. This result is defined only if the request mask (DL) had the read bit set and the call was successful.

Description:

This call is used to read/write the time slice and time count parameters of the specified process. If the write bit of the request mask is set then the current time slice and the current time count parameters are replaced by the specified new time slice. If the read bit was set in the request mask then the values of the time slice and time count parameters are returned. Note that time count is the process parameter which counts down, whereas the time slice parameter is used to initialize time count every time it reaches zero. If modified, the new value of time slice is put into effect immediately and thus affects the duration after which a rescheduling will be invoked due to the process exhausting its time slice.

PROC_STATUS

INT 69

Parameters:

BX: process-id (0 implies the calling process).

Results:

AL: completion code:

0 successful completion.

0F1H specified process does not exist.

AH: the status of the process:

0 process running, not locked.

1 process running, locked.

2 process runnable.

3 process blocked.

BX: process-id of the specified process, not destroyed if call fails.

Description:

This call returns the status of the specified process.

PROC_LOCK

INT 70

Parameters:

none.

Results:

AL: completion code:

0 successful completion.

AH: undefined.

Description:

This call causes scheduling decisions to be suspended until a corresponding PROC_UNLOCK call is executed. A lock is said to be in effect for the duration of suspension. This call, in conjunction with PROC_UNLOCK, can be used to exclude other processes in critical sections. A process can nest locks up to 32K levels deep. To unlock the process, each PROC_LOCK call should be matched by a corresponding PROC_UNLOCK call - in a manner similar to open and close parentheses. Any attempt to exceed the nesting limit of 32K will result in an undefined action.

If a process having locks in effect executes a call that can potentially cause the process to block, then all locks in effect are removed. Examples of such calls are MLBX_RECV with a non-zero sleep count or a PROC_SLPCNT call that sets the sleep count of the calling process to non-zero.

PROC_UNLOCK

INT 71

Parameters:

none.

Results:

AL: completion code:

0 successful completion.

AH: undefined.

Description:

This call removes the effect of a single PROC_LOCK call. If, as the result of this call, no more locks are pending, then scheduling is resumed in a normal way. If any events occurred during the locked state that required a rescheduling, then rescheduling is invoked immediately. Every PROC_LOCK call should be matched by a corresponding PROC_UNLOCK call. A PROC_UNLOCK call is a no-op if no locks are in effect.

MLBX_CREATE

INT 72

Parameters:

BX: must be -1 (OFFFHH), else effect is undefined.

CX: initial number of null messages, must be a non-negative number.

Results:

AL: completion code:

0 successful completion.

0E0H failed, maximum number of mailboxes allowed already exists.

0E2H failed, less than zero number of initial null messages.

AH: undefined.

BX: id of the new mailbox if call successful, otherwise undefined.

Description:

This call creates a new mailbox and returns its id. The specified number of null messages are enqueued in the mailbox. Note that if the mailbox is being used as a semaphore, then this allows creating a semaphore with a specified initial count. The process and regular message queues are always empty when initialized.

MLBX_DELETE

INT 73

Parameters:**BX:** mailbox-id.**Results:****AL:** completion code:

0 successful deletion.

0E1H failed, specified mailbox does not exist.

AH: undefined.**BX:** undefined, not destroyed if the call fails.**Description:**

The specified mailbox is deleted. If any processes are blocked on this mailbox, then they are unblocked and resumed with the appropriate error code. Any unreceived messages in the mailbox are lost. It is the programmer's responsibility to ensure that deleting a mailbox has no harmful effects. The user should not delete a system mailbox.

MLBX_SEND

INT 74

Parameters:

- BX:** mailbox-id.
- SI:** offset part of the message address. OFFFFH specifies a null message.
- ES:** segment part of the message address. This must be 0, or effect is undefined.

Results:

- AL:** completion code:
- 0 successful completion.
 - 0E1H failed, specified mailbox does not exist.
 - 0E4H failed, invalid request for a system mailbox.
 - 0E5H failed, improper buffer (message buffer segment not 0).
- AH:** undefined.

Description:

This call sends the specified message to the specified mailbox. If one or more processes are waiting in the mailbox then the first process is unblocked and resumed, having received this message. If a process is unblocked then a rescheduling is invoked immediately.

The message must lie entirely within the address range 0-OFFFFH. The first field of the message should be a 32-bit link field available for use by NX/101. If the specified mailbox is a system mailbox then the message must be formatted according to the specifications of the corresponding system process.

A regular message is appended at the end of all other regular messages in the mailbox but in front of all null messages, if any. A null message is appended at the end of all messages in the mailbox. If only null messages are used then this call is equivalent to a V operation on a semaphore.

MLBX_RECV

INT 75

Parameters:

- BX:** mailbox-id.
- CX:** sleep count, in ticks of 20 milliseconds. A value of -1 (OFFFHH) represents infinity.

Results:

- AL:** completion code:
- 0 successful.
 - 0E1H failed, specified mailbox does not exist.
 - 0C0H failed, specified sleep count exhausted.
 - 0E3H failed, the mailbox is being deleted.
- AH:** undefined.
- SI:** offset of the message address if the call is successful, otherwise undefined. A value of -1 (OFFFHH) indicates a null message.
- ES:** segment part of the message address if the call is successful, otherwise undefined. NX/101 always returns 0.

Description:

If the mailbox's message queue contains any messages, either regular or null, then this call returns the address of the first message in the queue.

If there are no pending messages and the sleep count is non-zero, then the calling process is blocked and appended at the end of the mailbox's process queue. The sleep count of the process is set to the specified value. When a message becomes available, the call returns its address, as above. If the sleep count of the blocked process counts down to zero, or is explicitly set to zero by a PROC_SLPcnt call, before it receives a message, then the process is unblocked and returns from this call with the error code 0C0H.

Note that if the sleep count was specified to be zero then the process effectively does not block and returns immediately with the error code 0C0H. By specifying the sleep count to be infinity, finite non-zero, or zero, a process can choose to wait forever, wait for a finite time interval, or not wait at all to receive a message.

If this call is made with a non-zero sleep count, then any locks in effect are canceled, regardless of any errors the call may return.

TIME_GET

INT 76

Parameters:

- CX:** number of 16-bit words of clock value to be returned.
- DI:** offset part of the memory buffer address to which the clock value will be copied.
- ES:** segment part of the memory buffer address to which the clock value will be copied.

Results:

- AL:** completion code:
 - 0 successful.
- AH:** undefined.
- CX:** number of words actually copied.

Description:

This call copies the specified number of 16-bit words of the clock value into the specified memory buffer. The least significant word of the clock occupies the lowest address of the buffer. If the specified number of words to be copied is more than the actual number of the words in the clock, then the extra buffer remains unused. The clock is a 64-bit counter that is incremented every 20 milliseconds. When the EXOS/101 is reset, the clock is initialized as zero.

TIME_SET

INT 77

Parameters:

CX: number of words to be written.

DI: offset part of the memory buffer address from which the new clock value will be copied.

ES: segment part of the memory buffer address from which the new clock value will be copied.

Results:

AL: completion code:

0 successful.

AH: undefined.

CX: number of words actually written.

Description:

This call copies the specified number of words from the specified buffer into the clock counter, starting from the least significant word. If the specified number of words to copy is greater than the number of words in the clock, then the remainder are not used. The clock is a 64-bit counter that is incremented every 20 milliseconds.

NET_MODE

INT 78

Parameters:

CL: options mask, which defines various controller options. Available options are defined by the following bit OR-able values:

10H alignment error - enables reception of packets even if the number of bits received is not a multiple of 8.

20H CRC error - enables reception of packets even if the CRC check fails.

80H net disable - disables the Ethernet controller so that packets are not received or transmitted on the Ethernet. However, transmit requests are still processed by NX/101, and to user processes appear to complete successfully if an on net mode is selected.

All other bits are undefined and must be 0. This parameter is required only if a write is requested.

DL: request mask:

01 write request bit.

02 read request bit.

Read and write can be requested simultaneously (DL = 03). Other bits in mask must be 0, or effects are undefined.

DH: the new mode of the Ethernet controller. Possible values are:

00H off net. Disconnect from the net.

01H on net, perfect filtering. Connect to net, perfect filter for multicast addresses.

02H on net, imperfect filtering. Connect to net, only hardware filter for multicast addresses.

03H on net, promiscuous mode. Connect to net, receive all packets.

This parameter is required only if a write is requested.

Results:

AL: completion code:

0 successful.

AH: undefined.

CL: options mask prior to this call. This result is defined only if the request mask (DL) had the read bit set.

DH: mode prior to this call. This result is defined only if the request mask (DL) had the read bit set.

Description:

This call is used to read/write the network controller mode and options mask parameters. If the write bit in the request mask (DL) is set, then the specified mode is written. Only the modes defined above should be used. Other modes are reserved for Excelan and their effects are not defined. The options mask defines the errors that are acceptable for the packets. If the read bit in the request mask is set then the mode and options mask of the controller prior to this call are returned.

NET_ADDR

INT 79

Parameters:

DL: request mask:

01 write request bit.

02 read request bit.

Read and write can be requested simultaneously (DL = 03). Other bits in mask must be 0, or effects are undefined.

DH: address slot number. Designates the address slot which this request will work on. This can be the physical address slot (253) or any multicast address slot (between 1 and the limit defined by configuration).

DI: offset part of the address of a six byte array. If a write is requested, then this array must contain the network address to be written.

ES: segment part of the address of a six byte array described above.

Results:

AL: completion code:

0 successful completion.

0D1H the specified slot does not exist or access is not permitted.

0D3H improper address. Multicast slots can only take multicast addresses and the physical slot can only take a physical address. Attempting to write the broadcast slot (number 255) results in this error.

AH: undefined.

DL: If bit 3 (mask value 8) is set, then the address slot contained a valid address prior to this call, otherwise the slot was empty. All other bits are undefined. This result is valid only if a read was requested.

Description:

This call is used to read/write an address in the specified address slot. If the write bit is set in the request mask, then the network address is copied into the specified slot from the array whose address is specified in DI, ES. If the read bit was set in the request mask then the network address in the specified address slot prior to this call is copied into the array whose address is specified in DI, ES. The address read is valid only if the slot was not empty prior to this call (DL). If a network address to be written is invalid, the write does not occur, and the

address in the slot prior to the call is preserved. Writing an address into a slot disables receive on that slot. The call NET_RECV must be explicitly used to enable receive on the slot.

Address slot 253 is reserved for the physical address and address slot 255 is reserved for the broadcast address. Thus the user can find the physical address by reading the address in slot 253.

NET_RECV

INT 80

Parameters:

DL: request mask:

01 write request bit.

02 read request bit.

04 enable receive bit.

Read and write can be requested simultaneously (DL = 03). Other bits in mask must be 0, or effects are undefined.

DH: address slot number. Designates the address slot which this request will work on. This can be the physical address slot (253), the broadcast slot (255), or any multicast address slot (between 1 and the limit defined by configuration).

Results:

AL: completion code:

0 successful completion.

0D1H the specified slot does not exist or access is not permitted.

0D2H the address slot is empty.

AH: undefined.

DL: If bit 2 (mask value 4) is set, then the receive was enabled for this slot prior to this call, otherwise it was disabled. All other bits are undefined. This result is defined only if read was requested.

Description:

This call is used to read/alter the receive status of an address slot. If the write bit is set in the request mask, then the receive is enabled or disabled depending on bit 2 of the request mask. If bit 2 (mask = 4) is set, then receive is enabled, otherwise it is disabled. If the read bit was set in the request mask then the receive status of the address slot prior to this call is returned.

NET_STSTCS

INT 81

Parameters:**DL:** request mask:

01 write request bit.

02 read request bit.

Read and write can be requested simultaneously (DL = 03). Other bits in mask must be 0, or effects are undefined.

CX: number of objects to be read/reset.**SI:** index into the statistics objects array.**DI:** offset part of the buffer address to which the statistics objects are to be copied.**ES:** segment part of the buffer address to which the statistics objects are to be copied.**Results:****AL:** completion code.

0: successful.

CX: the actual number of objects read/reset.**Description:**

This call reads/resets the statistics objects. Net statistics are an array of 32-bit objects, described in section 7.6. If the read bit is set in the request mask then the statistics objects starting at the index specified by SI are copied into the array specified by DI, ES. The number of objects to be copied is specified in CX. If the write bit is set in the request mask, then the number of objects specified by CX, starting at the index specified by SI, are reset to zero. The actual number of objects read/reset is returned in CX. If the index specified in SI is out of range, then no objects are read/reset.

MEM_READ

INT 82

Parameters:

CX: number of bytes to be read.
DX: high-order word of the address in system memory.
SI: low-order word of the address in system memory.
DI: offset part of the address in EXOS/101 memory.
ES: segment part of the address in EXOS/101 memory.

Results:

AL: completion code:
 0 successful.
AH: undefined.

Description:

This call copies the specified number of bytes from the specified address in system memory to the specified address in EXOS/101 memory. If the host data order conversion option is selected (see section 4.2), then any required conversion for the byte string data type will be done.

MEM_WRITE

INT 83

Parameters:

CX: number of bytes to be written.
DX: high-order word of the address in system memory.
SI: low-order word of the address in system memory.
DI: offset part of the address in EXOS/101 memory.
ES: segment part of the address in EXOS/101 memory.

Results:

AL: completion code:
 0 successful.
AH: undefined.

Description:

This call copies the specified number of bytes from the specified address in EXOS/101 memory to the specified address in system memory. If the host data order conversion option is selected (see section 4.2), then any required conversion for the byte string data type will be done.

CVT_WORD

INT 85

Parameters:

BX: word operand to be converted.

Results:

BX: converted word operand.

Description:

If the host data order conversion option (see section 4.2) is selected, then this call performs any required conversion for the word data type. It converts a word in the host system's native ordering to the 8088 CPU's native ordering, and vice versa. The only conversion relevant to this data type is byte swapping; its necessity is determined by the same test pattern which enables conversion for message queue data structures. CVT_WORD assumes that any conversion required for the byte string data type has already been performed on the operand. If the host data order conversion option is not selected, then the operand is returned without modification.

CVT_LWORD

INT 86

Parameters:

BX: first word of longword operand to be converted.

DX: second word of longword operand to be converted.

Results:

BX: first word of converted longword operand.

DX: second word of converted longword operand.

Description:

If the host data order conversion option (see section 4.2) is selected, then this call performs any required conversion for the longword data type. It converts a longword in the host system's native ordering to the 8088 CPU's native ordering, and vice versa. Possible conversions are word swapping, byte swapping, or both. Note that all possible conversions are symmetrical, and reflexive. Therefore the order of first and second word parameters to this call is not important, as long as user software treats the result consistently.

Necessary conversions are determined by the same test pattern which enables conversion for message queue data structures. CVT_LWORD assumes that any conversion required for the byte string data type has already been performed on the operand. If the host data order conversion option is not selected, then the operand is returned without modification.

VERSION

INT 84

Parameters:

none.

Results:

AL: always 0.

AH: undefined.

CX: version of NX/101.

DX: version of the EXOS/101 hardware.

Description:

This call returns the version number of the EXOS/101 hardware and NX/101. Version numbers have the form X.Y, where the lower byte (CL or DL) contains the ASCII value of X and the higher byte (CH or DH) contains the ASCII value of Y.

10. INITIALIZING AND DOWN-LOADING FROM THE ETHERNET

The EXOS/101 can be configured and down-loaded from its Ethernet interface in a manner quite similar to initialization by a host processor. This permits its use as a system master where the system's design does not include another CPU card, or it provides a convenient way to bootstrap diskless workstations. NX/101 firmware includes a simple protocol which supports the network bootstrap function. This section describes the network bootstrap protocol and provides information sufficient to implement a corresponding bootstrap server.

Network bootstrap is initiated either by a jumper option (see section 11) upon reset, or explicitly by a host system during configuration (see section 4.4.4). In either case, the EXOS/101:

- 1) finds a network bootstrap server on the Ethernet.
- 2) builds up a session with the boot server.
- 3) processes commands, including configuration and software down-load, received from the boot server.
- 4) executes the down-loaded code.

The network bootstrap protocol is based on request and reply messages which are encapsulated in standard Ethernet packets. The Ethernet type field identifies net boot packets as belonging to an Excelan protocol type. Another sub-type field designates the EXOS/101 network bootstrap protocol specifically.

10.1. Network Bootstrap Protocol Description

Figure 10-1 shows a state diagram of the network bootstrap protocol, both for client (the EXOS/101) and for boot server. In this diagram, states are represented as capitalized names enclosed in circles. State transitions appear as solid lines, with an arrow at one end to indicate the direction of the transition. Ethernet messages are shown as broken double lines, with the name of the message imbedded. An arrow at one end indicates the direction of transmission. Reception of an Ethernet message defines an event, and usually triggers a state transition. Note that transmission by one party does not guarantee reception by the other.

Whenever the event driving a state transition is a timeout, the line includes a C language expression in parentheses, such as "(f<FR)," or "(f>=FR)." The lower-case identifiers to the left in such an expression are counters, and refer to the number of timeouts of this kind which have occurred so far, while the upper-case constants to the right refer to the maximum number of retries permitted for this timeout. When a timeout occurs, the state transition taken will be that for which the expression is true. The counters are initialized and modified according to specific events, usually packet transmission or arrival. The appropriate action is shown as a C language statement enclosed in curly braces below the associated event. For example, "{f++}" increments the FIND request counter whenever the client transmits that message.

EXOS/101 states, shown to the left of the diagram, are as follows:

RESET denotes that the EXOS/101 has been reset, but has not yet attempted a network bootstrap.

FIND REPLY WAIT denotes that the EXOS/101 has sent one or more find request messages, and not yet received a reply to the most recent one.

SELECT REPLY WAIT denotes that the EXOS/101 has sent one or more SELECT request messages, and not yet received a reply to the most recent one.

COMMAND REQUEST WAIT denotes that the EXOS/101 has received a SELECT reply message, and is now awaiting a command request message from the selected boot server.

EXECUTE denotes that the EXOS/101 has received a valid execute request message, and sent the corresponding reply message. It now begins to execute the code which has presumably been down-loaded by the boot server.

ABORT denotes that the network bootstrap attempt has failed, after exhausting a specified number of retries (16 by default). The EXOS/101 displays an error code on the status LED until it is reset.

Boot server states for a straightforward implementation (shown to the right of the diagram) are as follows:

BOOT REQUEST WAIT denotes that the boot server is prepared to build a boot session with some EXOS/101 client. In this state, it responds both to find request and SELECT request messages.

COMMAND REPLY WAIT denotes that the boot server has received a SELECT request message, and sent back a SELECT reply message, thereby establishing a boot session with some EXOS/101 client. The boot server proceeds directly to send a command request message, and then awaits a command reply message from the client associated with this session.

State transitions occur only in response to some asynchronous event. In the network boot protocol, two basic types of event occur: arrival of a message on the Ethernet, or a timeout while waiting for some message. An exception is the START NETBOOT event, which actually encompasses two circumstances (neither of which involves Ethernet messages) that can initiate the network bootstrap procedure.

The network bootstrap protocol is based on three general types of request message. For each request message, the protocol defines a reply message whose format is identical. These message pairs are as follows:

- 1) The EXOS/101 broadcasts the FIND request message to discover the existence and address of bootstrap servers on the network. All bootstrap servers which receive this message send back a FIND reply message.

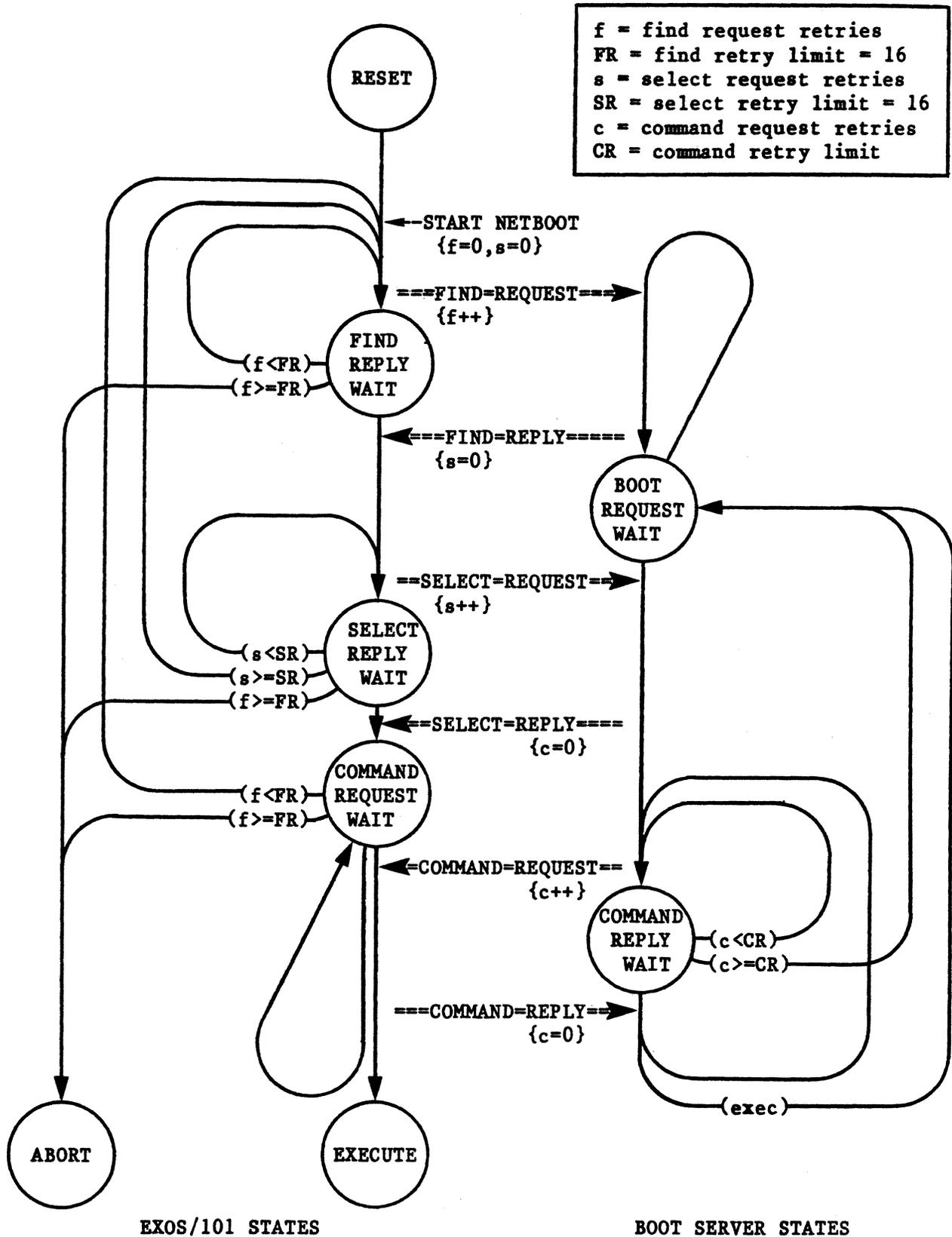


Figure 10-1: State Diagram of Network Bootstrap Protocol

EXOS/101: Initializing and Down-Loading from the Ethernet

- 2) The EXOS/101 sends the SELECT request message to the one bootstrap server which it wants to control the subsequent bootstrap process. The selected bootstrap server acknowledges its readiness to perform this role by sending back the SELECT reply message.
- 3) The bootstrap server can use several different COMMAND request messages to configure the EXOS/101, down-load code, up-load its memory contents, and begin execution. For each request, the EXOS/101 returns a COMMAND reply message to the bootstrap server.

A normal network bootstrap (where no packets are lost, all necessary resources are available, and nobody crashes) proceeds as follows, from the EXOS/101's point of view:

- 1) The EXOS/101 initiates the network bootstrap procedure upon either a hardware or software reset, if the net boot jumper is selected. If the net boot jumper is not selected, it can still initiate a network bootstrap upon initialization by the host system. In any case, it gets the ball rolling by broadcasting a FIND request message. This message contains:
 - a) the version number of the network bootstrap protocol which the EXOS/101 supports.
 - b) the number of buffers available on the EXOS/101 to receive incoming network bootstrap COMMAND request messages.
 - c) the length of the buffers described above.
 - d) the Ethernet address of the EXOS/101 (this is a separate field from the standard Ethernet source address field).
 - e) a message ID, which uniquely identifies the request message.
 - f) a timeout parameter, which tells the boot servers how long the EXOS/101 will wait for a reply message before re-trying or giving up.
 - g) a configuration message, which describes the current configuration of the EXOS/101. This is nearly identical to the configuration reply message returned during initialization by a host system (see section 4.4).

After sending the request message, the EXOS/101 enters the FIND REPLY WAIT state.

- 2) Before its FIND reply timeout expires, the EXOS/101 should receive a FIND reply message from at least one qualified bootstrap server. If more than one is received, the EXOS/101 selects the first to arrive, and discards subsequent FIND reply messages. This message provides the following information:

- a) the Ethernet address of the bootstrap server (as above, this is a separate field from the standard Ethernet source address field).
- b) a timeout parameter, which specifies how long the EXOS/101 should wait for the boot server's SELECT reply message.

Immediately upon receiving a legitimate FIND reply message, the EXOS/101 sends a SELECT request message to the bootstrap server whose address was contained in the reply message. This tells the designated bootstrap server that it is responsible for bootstrapping this EXOS/101 client. The SELECT request message contains exactly the same information as the FIND request message, except possibly for the timeout parameter. The EXOS/101 specifies its current effective timeout value in this field. After sending the SELECT request message, the EXOS/101 enters the SELECT REPLY WAIT state.

- 3) Before its SELECT reply timeout expires, the EXOS/101 should receive a SELECT reply message from the selected bootstrap server. This contains the same information as the FIND reply message, except possibly for the timeout parameter, which now tells the EXOS/101 how long to wait before giving up on receiving a COMMAND request message from the boot server. Reception of the SELECT reply message establishes a bootstrap session, and the EXOS/101 enters the COMMAND REQUEST WAIT state.
- 4) Before its COMMAND request timeout expires, the EXOS/101 should receive a COMMAND request message from the selected bootstrap server. When a command arrives, the EXOS/101 processes the command and returns a COMMAND reply message to the bootstrap server, informing it of the command's result. After sending the reply message, the EXOS/101 normally returns to the COMMAND REQUEST WAIT state. However, if the command was an EXECUTE request, the bootstrap session is terminated (as far as the EXOS/101 is concerned) and the EXOS/101 proceeds to execute the code which has presumably been down-loaded.

While the description above specifies exactly how the EXOS/101 will behave during a network bootstrap session, the bootstrap server's behavior is largely up to its implementor. The network bootstrap protocol is implemented with a typical bootstrap server model in mind, such as is shown in the state diagram. A real boot server might support more than one boot session simultaneously; the diagram shows only the context of a single boot session.

Note also that this diagram describes only the case where the EXOS/101 provides just one receive buffer for processing net boot commands. Therefore it assumes that only one command may be outstanding. Future releases of NX/101 may permit pipelined boot command processing by supplying multiple buffers. While this model for a boot server will still work when more buffers are available, it will not derive any performance advantage. At any rate, from the boot server's point of view, net boot proceeds as follows:

EXOS/101: Initializing and Down-Loading from the Ethernet

- 1) The bootstrap server starts in the BOOT REQUEST WAIT state, awaiting the arrival of either a FIND request message or a SELECT request message. Upon reception of a FIND request message, the boot server examines relevant information in this message, such as the protocol version. If the boot server decides that it can service the client which the request identifies, it sends back a FIND reply message to the address contained in the request. This message tells the client the boot server's address and how long a timeout it should use when waiting for subsequent messages from the boot server. The boot server then returns to the BOOT REQUEST WAIT state.
- 2) When the bootstrap server receives a SELECT request message, it records the information it will need to boot the client, and returns a SELECT reply message. Once again, this contains a timeout parameter which tells the client how long to wait for subsequent messages. At this point, a bootstrap session has been established, so far as the boot server is concerned.
- 3) After sending a SELECT reply message, the bootstrap server proceeds immediately to send COMMAND request messages to the client. After sending any COMMAND request message, the bootstrap server enters the COMMAND REPLY WAIT state.
- 4) Before its COMMAND reply timeout expires, the bootstrap server should receive a COMMAND reply message. It then sends the next COMMAND request message and re-enters the COMMAND REPLY WAIT state. However, if the COMMAND reply message was that of an execute request, then the bootstrap session is terminated and the boot server returns to the BOOT REQUEST WAIT state.

The description so far of the network bootstrap protocol has been simplified somewhat by ignoring considerations such as spurious messages or lost packets. However, these things can happen. Therefore, the protocol provides mechanisms which can accommodate errors during, and ensure completion of, the network bootstrap process.

Once the boot server's address is established, the EXOS/101 will ignore messages from other sources. Another general principle the EXOS/101 obeys is to ignore any message types it does not expect in its current state. For instance, COMMAND request messages will have no effect if the EXOS/101 is still in the SELECT REPLY WAIT state. A straightforward boot server implementation would also follow these rules.

The network bootstrap protocol uses a timeout/retry mechanism to recover from lost messages and various catastrophic circumstances. In any state where the EXOS/101 is waiting for some message to arrive, if the message does not arrive within some specified real-time interval (3000 milliseconds by default), the EXOS/101 will timeout. Depending on circumstance, it may then abort or retry, possibly entering a different state. The EXOS/101 maintains two counters which help determine the appropriate action. The FIND request counter is reset by the START NETBOOT event, and is incremented every time a FIND request message is transmitted. The SELECT request counter is reset when a FIND reply message is received, and is incremented every time a SELECT request message is transmitted. State transitions which occur on timeout events are described

below, according to the state before timeout:

FIND REPLY WAIT: When a timeout occurs, the EXOS/101 normally transmits another FIND request message and returns to the FIND REPLY WAIT state. However, if the FIND request counter shows that 16 FIND request messages have already been sent, then the net boot attempt is aborted and the EXOS/101 enters the ABORT state. The EXOS/101 will then display the appropriate error code on its status LED (see section 11). If the net boot was instigated by a host system, then the appropriate error code is also written into the configuration message's completion code field in host memory (see section 4.4.3).

SELECT REPLY WAIT: When a timeout occurs, the EXOS/101 normally transmits another SELECT request message and returns to the SELECT REPLY WAIT state. However, if the SELECT request counter shows that 16 SELECT request messages have already been sent, then the EXOS/101 transmits another FIND request message and returns to the FIND REPLY WAIT state. If the FIND request counter also shows that 16 FIND request messages have already been sent, then the net boot attempt is aborted, as above.

COMMAND REQUEST WAIT: When a timeout occurs, the EXOS/101 normally transmits another FIND request message and returns to the FIND REPLY WAIT state. However, if the FIND request counter shows that 16 FIND request messages have already been sent, then the net boot attempt is aborted, as described above.

Timeout processing in the bootstrap server is up to the implementor. In the typical implementation which the state diagram describes, only the COMMAND REPLY WAIT state can generate a timeout event. The timeout period, and the number of retries allowed, are dependent on the implementation. Typically, the timeout period multiplied by the number of retries allowed should not greatly exceed the EXOS/101's COMMAND REQUEST WAIT timeout (which can be specified by the boot server in the SELECT reply message).

During a network bootstrap attempt, it is possible that either the client or server could receive messages generated during some prior network bootstrap attempt gone awry. For instance, if the SELECT reply message is lost, then a boot server would still assume that a session had been established, and would persist in retrying on its first COMMAND request message. Meanwhile, the EXOS/101 might have established a new boot session. Some means is needed to distinguish between messages belonging to the legitimate boot session and the defunct boot session. For this purpose, the network bootstrap protocol supports the concept of message IDs and, once a session is established, session IDs.

The EXOS/101 generates a message ID field for both FIND request and SELECT request messages. This ID guards the EXOS/101 against spurious message reception up to the point that a network bootstrap session is established. The EXOS/101's message ID generation algorithm guarantees that it will be unique in each and every message from the time the board is reset until it is reset again. Furthermore, the field contains a random component which makes ID collision very unlikely even after a reset occurs. As described above, the boot server is expected to copy the message ID from FIND and SELECT request messages into their corresponding reply messages.

When the boot server establishes a session (by returning the SELECT reply message), it is responsible for creating a unique 12-byte session ID value, which it passes to the EXOS/101 in the SELECT reply message. In all subsequent COMMAND request messages, the boot server should write this value into the first 12 bytes of the 16-byte message ID field. When the EXOS/101 receives a COMMAND request message, it ignores it unless the first 12 bytes of the message ID field agree with the value it received in the SELECT reply message. When the EXOS/101 prepares a reply message, it copies the entire message ID field from the corresponding request message. The remaining 4 bytes at the end of the ID field can be used for any purpose which suits the boot server - typically message serialization.

10.2. Data Transmission Order

This section defines the order of transmission for data objects which are known to the network bootstrap protocol implemented by NX/101. Network bootstrap servers must obey these conventions when transmitting messages to the EXOS/101, and should observe them when interpreting messages received from the EXOS/101.

The fields defined by the Ethernet specification for the standard data link layer frame format (destination address, source address, type, and frame check sequence) are, of course, transmitted in their standard order. The Ethernet specification also defines how the contents of a packet's data field (which contains all network bootstrap message contents) are to be transmitted, but only in terms of bit significance. For each byte in a packet's data field, the Least Significant Bit (LSB) is transmitted first, and the Most Significant Bit (MSG) last.

The byte ordering of multi-byte data objects in network bootstrap messages is defined solely by the network bootstrap protocol. This follows a simple rule. All data objects are transmitted as though stored in memory according to the 8088 CPU's native order, and then transmitted one byte at a time, starting at the lower memory address. This is the transmission order which naturally occurs when the network bootstrap protocol is implemented on the EXOS/101. When implementing a bootstrap server based on a different CPU architecture, programmers should be careful to observe this ordering.

Note that the EXOS/101 host data order conversion option does not apply to the contents of network bootstrap messages. However, the option may be enabled as usual by the CONFIGURE request message. Simply set up the test pattern field as it would have been written by the system being bootstrapped. Data conversion will then work on all messages passed between the client EXOS/101 and its host processor. The rest of the CONFIGURE request message, and all other messages, will still be interpreted according to 8088 data ordering.

10.3. Network Bootstrap Protocol Message Header

Network bootstrap request and reply messages share a common header format, shown in figure 10-2. The following paragraphs describe its individual fields in detail.

#	Length	Offset	Field Name	Request	Reply
1)	2	0	Subtype	1080H	preserved
2)	16	2	: Message ID	: see text	: see text
3)	1	18	Request Code	see text	preserved
4)	1	19	Reply Code	undefined	see text
5)	2	20	Message Length	see text	see text
6)	n	22	: Request-Specific Fields...	: see text	: see text
			<-----1 byte----->		

Figure 10-2: Network Bootstrap Protocol Request/Reply Message Header

10.3.1. Subtype Field

The subtype field identifies specific Excelan protocol types. The network bootstrap protocol's type is 0; all request and reply messages must contain this value.

10.3.2. Message ID Field

The message ID field is used before a session is established to associate request messages with the corresponding reply messages. The EXOS/101 generates unique message ID numbers for the FIND and SELECT request messages, and the boot server simply copies these numbers into the corresponding reply messages. Once a session is established, this field identifies all request and reply messages as belonging to that session, and can be used to serialize messages. The boot server generates the session ID number used in all subsequent COMMAND request and reply messages. The following sections will explain this field in more detail.

10.3.3. Request Code Field

The request code field identifies the particular request or reply contained in a network bootstrap message. Values are as follows:

0	DOWNLOAD
1	UPLOAD
2	EXECUTE
3	CONFIGURE
4	FIND
5	SELECT

The same code is used for both request and reply messages. They are distinguished from each other by context.

10.3.4. Reply Code Field

The reply code field returns the result of a request. It must be 0 in request messages. Its meaning in reply messages will be explained in the individual message descriptions below.

10.3.5. Message Length Field

The message length field defines the number of bytes beyond its own position in the Ethernet packet containing the request or reply message. As usual, this should not include the CRC field's length.

10.3.6. Request-Specific Fields

Beyond the message length field, the remainder of each request message is defined according to the purpose of the request. These fields are described below, in the individual message descriptions.

10.4. Message Encapsulation

Request and reply messages are encapsulated in the data field of a standard Ethernet packet, as shown in figure 10-3.

The EXOS/101 places the physical address of a boot server in the destination address field, except in the FIND request message, where it contains the Ethernet broadcast address. The boot server should always place the physical address of a client EXOS/101 in the destination address field.

The source address field always contains the physical address of the party which sent the message.

ETHERNET PACKET

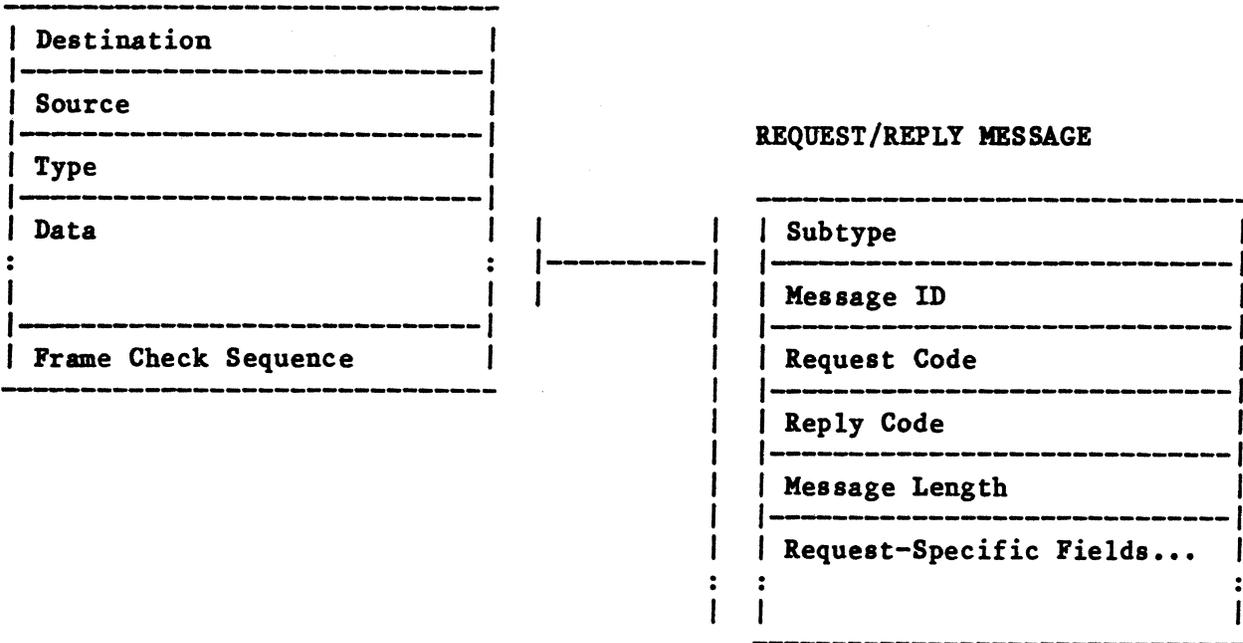


Figure 10-3: Encapsulation of Request/Reply Message

The type field should always contain the Excelan protocol type, which in Ethernet parlance is:

80-10

The value above is given in hexadecimal notation, and should be transmitted left-most byte first. On the EXOS/101 itself, this is equivalent to storing the 16-bit value 1080H in the 8088 CPU's native order.

The following sections describe the individual request and reply messages, including a detailed description of the data fields unique to each request. The diagrams for these messages do not show the individual Ethernet fields or the standard message header fields. Offset addresses shown for the messages are calculated from the beginning of the standard message header (at the subtype field).

10.5. FIND and SELECT Request/Reply Messages

The FIND and SELECT request messages are described together here because their format, shown in figure 10-4, is identical. The EXOS/101 broadcasts the FIND request message to identify bootstrap servers, which return a FIND reply message to the client's physical address. The EXOS/101 then sends the SELECT request message to the physical address of a boot server, telling it to bootstrap the EXOS/101. The boot server acknowledges this with a SELECT reply message. The following paragraphs describe the individual fields in detail.

EXOS/101: Initializing and Down-Loading from the Ethernet

Unless otherwise stated, each field's function is identical in FIND and SELECT messages.

#	Length	Offset	Field Name	Request	Reply
1)	22	0	: Standard Message Header Fields : :	see text	see text
2)	2	22	Protocol Version	1	preserved
3)	2	24	Number of Buffers	1	preserved
4)	2	26	Buffer Length	512	preserved
5)	6	28	Station ID	see text	see text
6)	12	34	: Session ID : :	undefined	see text
7)	2	46	Receive Wait Timeout	see text	see text
8)	80	48	: Configuration Message : : (in request messages only) :	see text	-
			<-----1 byte----->		

Figure 10-4: Network Bootstrap FIND/SELECT Request/Reply Message

10.5.1. Standard Message Header Fields

The EXOS/101 writes a unique value into the message ID field in each request message. The boot server should return this same value in the reply message, enabling the EXOS/101 to associate the two.

The request code field's value in the FIND request is 4, in the SELECT request 5. The boot server should return the same value in the reply message.

The reply code field should be 0 in both request and reply messages.

The message length field contains the value 106 in the request messages. Its

value in the reply message should be 26.

10.5.2. Protocol Version Field

The protocol version field contains the revision level of the network bootstrap protocol supported by the EXOS/101. Boot servers can examine this field to check that they are compatible with the client's version. It is interpreted as a simple 16-bit numeric value. The current protocol version is 1. The boot server should preserve this value in the reply message.

10.5.3. Number of Buffers Field

The number of buffers field tells the boot server how many buffers the client EXOS/101 provides for processing COMMAND requests. This determines how many outstanding requests the boot server should allow at any time. Its current value is 1. The boot server should preserve this value in the reply message.

10.5.4. Buffer Length Field

The buffer length field specifies the length of the EXOS/101's receive buffer. This determines the maximum size COMMAND request packets the EXOS/101 can receive, excluding the 4-byte CRC field. Its current value is 508 bytes. The boot server should preserve this value in the reply message.

10.5.5. Station ID Field

The station ID field contains the physical address, in standard Ethernet format, of the party to which a message pertains. While this is normally the same as a packet's source address field, this is not necessarily the case. A bootstrap server might place a different boot server's address in this field in order to "hand off" a boot session. The EXOS/101 examines this field in FIND and SELECT reply messages to determine the boot server's physical address. The boot server should examine this field to determine the client's physical address, as well. The EXOS/101 will always place its effective physical address in this field.

10.5.6. Session ID Field

The session ID field is undefined in the FIND request and reply messages. It is also undefined in the SELECT request message. In the SELECT reply message, the boot server should return a unique value in this field which identifies the boot session just established. The EXOS/101 will then accept COMMAND request messages only if the first 12 bytes of their message ID field matches this value.

10.5.7. Receive Wait Timeout Field

The receive wait timeout field is used to negotiate the timeout interval which the EXOS/101 observes when waiting for some message from the boot server. It is specified in milliseconds, but the EXOS/101 will round it up to the next 20-millisecond interval if it is not an even multiple of 20. In the FIND request message, the EXOS/101 declares the current value, which is 3000 milliseconds by default. The default value is in force after a reset, and is reinstated whenever the EXOS/101 performs a FIND request retry. Therefore the

EXOS/101 will timeout and retry if it has not received a FIND reply message within 3 seconds after sending the FIND request.

The boot server can specify a different value in the FIND reply message, and the EXOS/101 will copy this value (subject to rounding) into the SELECT request message. In the SELECT reply message, the boot server can once again specify a different value. In either reply message, the value OFFH selects the current value. If the value specified is 0, then the EXOS/101 will not timeout, but will wait indefinitely. This is useful for debugging purposes.

10.5.8. Configuration Message Field

The configuration message field is defined only for the FIND and SELECT request messages. The reply messages do not include this field and the boot server need not allocate space for it in the message length field. Its format is exactly identical to the configuration message described in section 4.4; it should be interpreted as though it were a configuration reply message. It describes the current configuration of the EXOS/101, which will express all the default values if the board has just been reset. If the board has been configured previously (which may occur if initialized by a host system) then it will reflect any modifications made since reset time.

10.6. DOWNLOAD Request/Reply Message

The bootstrap server can use the DOWNLOAD request message to down-load code and data to the EXOS/101's RAM. Any area of memory normally available to the user can be used. Figure 10-5 shows the format of the request message, and the following paragraphs describe its individual fields in detail.

10.6.1. Standard Message Header Fields

The boot server should write the session ID into the first 12 bytes of the message ID field in each request message. The remaining 4 bytes may be used for any purpose which suits the boot server. In the reply message, the EXOS/101 will preserve this entire field's value.

The request code field's value for the DOWNLOAD request is 0. The EXOS/101 returns the same value in the reply message.

The reply code field should be 0 in the request message. In the reply message, it reports the status of the DOWNLOAD request.

0 successful completion.

A3H destination memory block overlaps the memory reserved for NX/101, no copy done.

AlH invalid request.

The message length field will depend on the length of the data field in the request message. Its value in the reply message is 10.

#	Length	Offset	Field Name	Request	Reply
1)	22	0	: Standard Message Header Fields :	see text	see text
2)	2	22	Load Length	see text	see text
3)	4	24	Reserved	zero	undefined
4)	4	28	EXOS Down-Load Address	see text	preserved
5)	n	32	: Data : (in request message only)	see text	-

|<-----1 byte----->|

Figure 10-5: Network Bootstrap DOWNLOAD Request/Reply Message

10.6.2. Load Length Field

The load length field specifies the length of the data field in the request message. In the reply message, this field returns the number of bytes actually down-loaded into EXOS/101 memory.

10.6.3. Reserved Field

The reserved field should contain zeros in the request message. Its value is undefined in the reply message.

10.6.4. EXOS Down-Load Address Field

The EXOS down-load address field specifies the address in EXOS/101 memory to which the data should be transferred. Note that, as with all addresses referring to locations in EXOS memory, this should be a segmented address in the 8086 style. Its value is preserved in the reply message.

10.6.5. Data Field

In the request message, the data field contains the data to be down-loaded. Given the current receive buffer size of 508 bytes, the maximum size of this field is 462 bytes. The data field is not defined in the reply message, nor should space be allocated for it there.

10.7. UPLOAD Request/Reply Message

The bootstrap server can use the UPLOAD request to read data from the EXOS/101's RAM. It is similar to the DOWNLOAD request, except that the data field is defined for the reply message instead of the request message. Figure 10-6 shows the format of the request message, and the following paragraphs describe its individual fields in detail.

#	Length	Offset	Field Name	Request	Reply
1)	22	0	: Standard Message Header Fields : : : : :	see text	see text
2)	2	22	Load Length	see text	see text
3)	4	24	Reserved	zero	undefined
4)	4	28	EXOS Up-load Address	see text	preserved
5)	n	32	: Data : : (in reply message only) :	-	see text

|<-----1 byte----->|

Figure 10-6: Network Bootstrap UPLOAD Request/Reply Message

10.7.1. Standard Message Header Fields

The boot server should write the session ID into the first 12 bytes of the message ID field in each request message. The remaining 4 bytes may be used for any purpose which suits the boot server. In the reply message, the EXOS/101 will preserve this entire field's value.

The request code field's value for the UPLOAD request is 1. The EXOS/101 returns the same value in the reply message.

The reply code field is should be 0 in the request message. In the reply message, it reports the status of the UPLOAD request:

- 0 successful completion.

A3H specified memory does not exist, no copy done.

A1H invalid request.

The message length field's value in the request message should be 10. Its value in the reply message will depend on the length of the data field.

10.7.2. Load Length Field

The load length field in the request message specifies the number of bytes to be read from the EXOS/101's memory. In the reply message, this field returns the number of bytes actually read from EXOS/101 memory.

10.7.3. Reserved Field

The reserved field should contain zeros in the request message. Its value in the reply message is undefined.

10.7.4. EXOS Up-load Address Field

The EXOS up-load address field in the request message specifies the address in EXOS/101 memory from which to read data. In the reply message, its value is preserved.

10.7.5. Data Field

The data field is not defined in the request message, nor should space be allocated for it there. In the reply message, the data field contains the data read from EXOS memory. As in the DOWNLOAD command, this is constrained by the current receive buffer size of 508 bytes; its maximum size is 462 bytes.

10.8. CONFIGURE Request/Reply Message

The bootstrap server can use the CONFIGURE request to modify the EXOS/101's configuration, just as the host would at initialization time (see section 4.4). Normally, a boot server performs configuration with its first COMMAND request message, before down-loading software; after configuration the contents of user memory on the EXOS/101 is not defined. However, configuration is not mandatory; if neglected, all configuration options will retain their current values, or the default values if the board has not been configured since reset. Figure 10-7 shows the format of the request message, and the following paragraphs describe its individual fields in detail.

10.8.1. Standard Message Header Fields

The boot server should write the session ID into the first 12 bytes of the message ID field in each request message. The remaining 4 bytes may be used for any purpose which suits the boot server. In the reply message, the EXOS/101 will preserve this entire field's value.

The request code field's value for the CONFIGURE request is 3. The EXOS/101 returns the same value in the reply message.

#	Length	Offset	Field Name	Request	Reply
1)	22	0	: Standard Message Header Fields : : -----	see text	see text
2)	80	22	: Configuration Message : : (in request messages only) :	see text	see text
<-----1 byte----->					

Figure 10-7: Network Bootstrap CONFIGURE Request/Reply Message

The reply code field should be 0 in the request message. In the reply message, its value is the same as the configuration message's Completion Code field.

The message length field's value in the request message should be 80. This value is preserved in the reply message.

10.8.2. Configuration Message Field

The configuration message field is exactly equivalent to the configuration message described in section 4.4. There are some slight semantic differences which apply to net boot mode. For instance, in the request message, the number of hosts field may be 0. If so, then all the following fields, which specify host message queue parameters, are undefined. The EXOS operation mode field must always be set to 2, for net boot mode. In the reply message, it will always return this value.

10.9. EXECUTE Request/Reply Message

The boot server can use the EXECUTE request message to start execution of code it has down-loaded to the EXOS/101. Once the EXOS/101 receives this command, it will ignore all network bootstrap type packets. The initial process runs exactly the same as one initialized by a host system (see section 4.8). Figure 10-8 shows the format of the EXECUTE request/reply message, and the following paragraphs explain its individual fields in detail.

10.9.1. Standard Message Header Fields

The boot server should write the session ID into the first 12 bytes of the message ID field in each request message. The remaining 4 bytes may be used for any purpose which suits the boot server. In the reply message, the EXOS/101 will preserve this entire field's value.

The request code field's value for the EXECUTE request is 2. The EXOS/101 returns the same value in the reply message.

The reply code field should be 0 in the request message. In the reply

#	Length	Offset	Field Name	Request	Reply
1)	22	0	: Standard Message Header Fields : :	see text	see text
2)	4	22	Starting Address	see text	preserved
<-----1 byte----->					

Figure 10-8: Network Bootstrap EXECUTE Request/Reply Message

message, it reports the status of the EXECUTE request:

- 0 successful completion.
- A1H invalid request.
- A2H invalid starting address.

The message length field's value in both the request and reply messages should be 4.

10.9.2. Starting Address Field

The starting address field specifies the initial value of the initial process's program counter. Its value is preserved in the reply message.

11. HARDWARE REFERENCE

Most hardware-dependent aspects of EXOS/101 implementation are hidden by NX/101, ensuring that high-level software written for the EXOS/101 will be portable to future products. This section provides all necessary hardware interface and configuration information. Theory of operation is deliberately omitted.

11.1. Access to EXOS/101 Components

Appendix A shows the EXOS/101's layout, and the locations of accessible components. For development purposes, the following components are socketed:

8088 CPU
16K EPROM

The EXOS/101 provides several jumpers to select addresses and options. Figure 11-1 is a quick reference to jumper functions, by number. Jumpers installed as shipped from the factory are marked with an asterisk in this table. Most jumpers are located in two blocks near the bottom center of the board. Figure 11-2 shows relative locations and functions for these jumpers. Subsequent sections explain the jumpers in more detail.

The EXOS/101 includes three Light Emitting Diodes (LEDs) to communicate status information. These are located in adjacent positions at the top left side of the board, seen from the component side, and can easily be seen while the board is installed. Figure 11-3 briefly shows their individual locations and functions. Subsequent sections explain the LEDs in more detail.

11.2. Multibus Interface

The EXOS/101 Ethernet Front-End Processor is built on a single 12" by 6.75" Multibus board. It presents one TTL (LS) load on the Multibus.

11.2.1. Multibus Compliance

The EXOS/101 conforms to Multibus specifications as an 8-bit bus master. IEEE 796 compliance is MASTER D8 M24 I16 V0 L:

8-bit transfers.
24-bit addressing.
non-bus vectored interrupts.

11.2.2. Multibus Memory Access

The EXOS/101 can generate either 20 or 24-bit memory addresses, to access either 1 Mbyte, or 16 Mbytes, of Multibus memory. Jumper JU29 selects 24-bit addressing. If not installed, the 4 highest order address bits (on the J2 connector) are tri-stated. When enabled, these bits are driven dynamically by NX/101 firmware. As shipped from the factory, JU29 is installed.

Note that the EXOS/101's own memory is not accessible from the Multibus.

<u>jumper</u>	<u>function (when jumper is installed)</u>
JU1	I/O address bit 8 = 1 (JU16 not installed)
JU2	I/O address bit 1 = 1
JU3	I/O address bit 9 = 1 (JU16 not installed)
JU4	I/O address bit 2 = 1
JU5	I/O address bit 10 = 1 (JU16 not installed)
JU6	I/O address bit 3 = 1
JU7	I/O address bit 11 = 1 (JU16 not installed)
JU8	I/O address bit 4 = 1
JU9	I/O address bit 12 = 1 (JU16 not installed)
JU10	I/O address bit 5 = 1
JU11	I/O address bit 13 = 1 (JU16 not installed)
JU12	I/O address bit 6 = 1
JU13	I/O address bit 14 = 1 (JU16 not installed)
JU14	I/O address bit 7 = 1
JU15	I/O address bit 15 = 1 (JU16 not installed)
JU16	select 8-bit I/O address
JU17	select interrupt level 7 (lowest priority)
JU18	select interrupt level 6
JU19 *	select interrupt level 5
JU20	select interrupt level 4
JU21	select interrupt level 3
JU22	select interrupt level 2
JU23	select interrupt level 1
JU24	select interrupt level 0 (highest priority)
JU25	reserved, do not install
JU26	network bootstrap
JU27	test for upper 64K RAM
JU29 *	enable 24-bit addressing
JU30 *	connect BPRO/ to Multibus interface

Figure 11-1: Quick Reference to Jumper Options

11.2.3. Multibus I/O Access

The EXOS/101 can access the full 64K Multibus I/O address space. However, it does not normally generate any I/O commands, unless requested by user software.

The EXOS/101 presents two read/write I/O ports to the Multibus. Their functions are documented in section 4.1. Port A's address is fully jumper-selectable, at any even address. The port address can be either 8 or 16 bits in length. Port B's address is the address of port A plus 1. (Note that 68000 CPU boards such as the SUN design typically invert the least significant address bit, so that ports A and B are logically reversed as seen by host system software.)

For jumper functions and locations, see figures 11-1 and 11-2. An installed I/O address jumper selects a 1 in its corresponding address bit position. JU16 selects 8-bit port addressing when installed; otherwise the port address

Port Address Bit 8	-----	JU1	JU2	-----	Port Address Bit 1
Port Address Bit 9	-----	JU3	JU4	-----	Port Address Bit 2
Port Address Bit 10	-----	JU5	JU6	-----	Port Address Bit 3
Port Address Bit 11	-----	JU7	JU8	-----	Port Address Bit 4
Port Address Bit 12	-----	JU9	JU10	-----	Port Address Bit 5
Port Address Bit 13	-----	JU11	JU12	-----	Port Address Bit 6
Port Address Bit 14	-----	JU13	JU14	-----	Port Address Bit 7
Port Address Bit 15	-----	JU15	JU16	-----	Select 8-bit Port Address

Interrupt Level 7	-----	JU17			
Interrupt Level 6	-----	JU18			
Interrupt Level 5	-----	JU19			
Interrupt Level 4	-----	JU20			
Interrupt Level 3	-----	JU21			
Interrupt Level 2	-----	JU22	JU25	-----	Reserved
Interrupt Level 1	-----	JU23	JU26	-----	Down-Load from Ethernet
Interrupt Level 0	-----	JU24	JU27	-----	Optional RAM Installed

Figure 11-2: Port Address, Interrupt Level, and NX/101 Option Jumpers

is 16 bits. Jumpers for port address bits 8 through 15 are ignored if JU16 is installed. Address bit 0 is not selectable - it is always 0 for port A, and 1 for port B. As shipped from the factory, no port address jumpers are installed. Consequently the port addresses are 16 bits long: 0000H for port A, and 0001H for port B.

11.2.4. Multibus Interrupt Access

The EXOS/101 can assert non-bus vectored interrupts on the Multibus. Interrupt priority is jumper-selectable in the range from INTO to INT7. For jumper functions and locations, see figures 11-1 and 11-2. An installed jumper selects the corresponding interrupt level. Only one interrupt level jumper

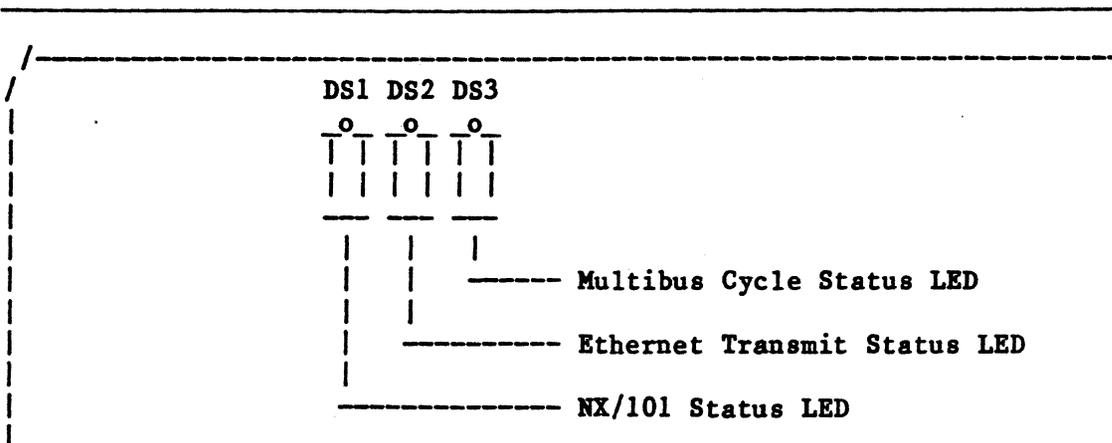


Figure 11-3: Quick Reference to Status LEDs

should be installed, or malfunction will result. As shipped from the factory, jumper JU19 is installed. Consequently interrupt level 5 is selected.

The EXOS/101 can also be initialized to generate memory-mapped or I/O mapped interrupts to the host. The host interrupts the EXOS/101 by writing to an I/O port (see section 4.1).

11.2.5. Multibus Priority Resolution

As a bus master, the EXOS/101 requests and releases the bus for each command. It executes the command immediately upon obtaining the bus, and releases the bus immediately upon the command's completion. Therefore its bus load is dependent only on the performance of the Multibus slave being accessed (typically host memory).

The EXOS/101 is compatible with either parallel or serial priority resolution schemes. Some Multibus implementations require the disconnection of the BPRO/line when parallel resolution is employed. Jumper JU30 on the EXOS/101 connects this signal to the bus interface, and may be removed if required. As shipped from the factory, JU30 is installed.

11.2.6. Multibus Cycle Status LED

The Light Emitting Diode (LED) in position DS3 on the EXOS/101 indicates that a multibus cycle is in progress, when lit. If lit steadily, then the EXOS/101 has probably attempted to access a non-existent or bad memory address on the Multibus. In general, this condition points toward a user software bug.

11.3. Ethernet Interface

Integrated with a standard Ethernet transceiver, the EXOS/101 performs all specified Ethernet physical and link level functions.

11.3.1. Ethernet Compliance

The EXOS/101 conforms fully to Ethernet specification, version 1.0, published September 30, 1980, by DEC, Intel, and Xerox.

11.3.2. Ethernet Functions

Functions implemented on the EXOS/101 board include:

serial/parallel and parallel/serial conversion.

physical and multicast address recognition.

packet framing and unframing.

Manchester encoding and decoding.

preamble generation and removal.

carrier sense and deference.

collision detection and enforcement.

backoff and retry timing.

frame check sequence (CRC) generation and verification.

alignment and length error detection and handling.

In addition to the standard Ethernet functions, the EXOS/101 implements a Time Domain Reflectometer with 100 ns resolution. Its measurements are included among the network statistics maintained by the NX/101 kernel.

11.3.3. Ethernet Address Recognition

The EXOS/101 recognizes physical, multicast, and broadcast addresses without user software intervention. A very efficient multicast address filter, implemented in hardware, greatly reduces the overhead of multicast address recognition. The multicast address filter can be disabled, so that all multicast addresses are accepted. The EXOS/101 also provides a promiscuous mode, in which it accepts all addresses.

Each EXOS/101 board has a unique 48-bit Ethernet address, stored in EPROM. This is the board's physical address by default, but the effective physical address resides in RAM, and may be modified by user software.

11.3.4. Ethernet Operation Timing

The EXOS/101 can receive successive frames with minimum interframe spacing (9.6 microseconds). It can also receive immediately after transmitting, or vice versa, with minimum interframe spacing, and without losing data.

11.3.5. Ethernet Packet Buffering

Under NX/101 firmware control, the EXOS/101 can buffer an arbitrary number of both receive and transmit packets. The actual number of available buffers depends on application criteria. User software can select both buffer size and location, anywhere between 01000H and 0FFFFH in the EXOS/101's dual-ported memory.

Ethernet controller hardware can chain up to 32 receive packet buffers, and receive as many packets, without CPU intervention. Transmit packets are chained by NX/101 firmware, and transmitted with minimal delay.

11.3.6. Ethernet Error Handling

The EXOS/101 can be selectively enabled to receive packets normally rejected due to CRC and alignment errors.

11.3.7. Ethernet Transmit Status LED

The EXOS/101 lights an LED at position DS2 while transmitting on the Ethernet.

11.3.8. Ethernet Transceiver Connector

The EXOS/101 board's Ethernet connector is a 16-pin IDH type which mates with a 16 pin IDC type connector. Pinouts are defined as per Ethernet specifications. The connectors are keyed, and pin number 1 can also be identified by an arrow on the connector. Note that it is still possible to insert the connector backwards. In order to ground the transceiver cable shield, pin number 1 must be connected to the host system chassis ground. A terminal connected to pin 1 is provided on the board for that purpose.

11.4. On-Board Processing Capabilities

The EXOS/101 is designed to facilitate the implementation of higher level communications protocols on its own processor. The major elements of this front-end processor are:

an 8 MHz 8088 CPU, clock speed 6.67 MHz.

64K of dual-ported RAM, 60K available for user software.

optionally, an addition 64K of single-ported RAM.

NX/101 OS kernel, residing in 16-Kbyte EPROM.

Access to RAM is subject to wait states; net effective throughput is equivalent to an 8088 running at 5 MHz or faster, without wait states. Access to EPROM does not incur any wait states.

The NX/101 kernel provides a real-time, multi-tasking environment for the implementation of higher level protocols on the EXOS/101. It is supported by clock timer and interrupt controller chips. NX/101 implements consistent and portable access methods for the Ethernet and host interfaces. In addition, it executes self-diagnostics, and can optionally drive the EXOS/101 as an

intelligent link level controller, in which case the user is not required to down-load protocol software.

11.5. Firmware Configuration Options

Jumpers JU25, JU26, and JU27 select NX/101 firmware options as follows:

- JU25 reserved for Excelan, must not be installed.
- JU26 if installed, the EXOS/101 will attempt to down-load software from the Ethernet after self-test is complete. If not installed, the EXOS/101 will await initialization from the host after self-test is complete.
- JU27 will cause self-diagnostics to abort initialization, and display an error code, if the upper 64K of RAM is not installed, or mal-functions.

11.6. Self-Test Operation

When the EXOS/101 is reset by the Multibus INIT/ line or by host software (see section 4.3), NX/101 firmware runs comprehensive diagnostic tests on EXOS/101 components. These tests complete within 2 seconds, whereupon the board is ready for configuration. If the tests fail, this is reported to the host via an I/O port (see section 4.1).

11.6.1. NX/101 Status LED

Test progress and status are also reported via an LED at position DS1. On EXOS/101 reset this LED is lit, and remains lit constantly while self tests are in progress. When self tests are complete, the LED flashes evenly until the EXOS/101 is initialized by the host or from the Ethernet. After initialization, LED DS1 is turned off.

If diagnostics indicate a hardware problem, then the LED will be lit constantly, or communicate an error code by flashing long and short pulses. Software errors during the process of configuration can also result in an error code display. Error codes are 8-bit numbers, and are presented bit-by-bit, starting with the most significant bit. A long pulse is a 1 bit, and a short pulse is a 0 bit. The error code is continuously repeated, with a pause in between to demarcate the starting point. Figure 11-4 specifies all defined error codes for the EXOS/101.

11.7. Power Requirements

<u>Hex Code</u>	<u>Pulse Code</u>	<u>Explanation of Error Code</u>
A0H	-.-.	invalid address for configuration message.
A4H	-.-. .-..	invalid operation mode parameter.
A5H	-.-. .-.-	invalid host data format test pattern.
A7H	-.-. .---	invalid configuration message format.
A8H	-.-. -...	invalid movable data block parameter.
A9H	-.-. -.-	invalid number of processes parameter.
AAH	-.-. -.-.	invalid number of mailboxes parameter.
ABH	-.-. -.-	invalid number of address slots parameter.
ACH	-.-. ---.	invalid number of hosts parameter.
ADH	-.-. ---	invalid host queue parameter.
AEH	-.-. ----.	improper objects allocation.
AFH	-.-. ----	net boot failed.
BOH	-.---	checksum on NX/101 EPROM failed.
B1H	-.--- ...-	memory test failed for 0-64K.
B2H	-.--- ..-	memory test failed for 64K-128K.
B3H	-.--- ..--	counter test failed.
B4H	-.--- .-..	interrupts test failed.
B5H	-.--- .-.-	transmission test failed.
B6H	-.--- .--.	receive test failed.
B7H	-.--- .---	local loopback data path test failed.
B8H	-.--- -...	CRC test failed.
B9H	-.--- -.-	checksum on physical address EPROM failed.
BAH	-.--- -.-.	system error.

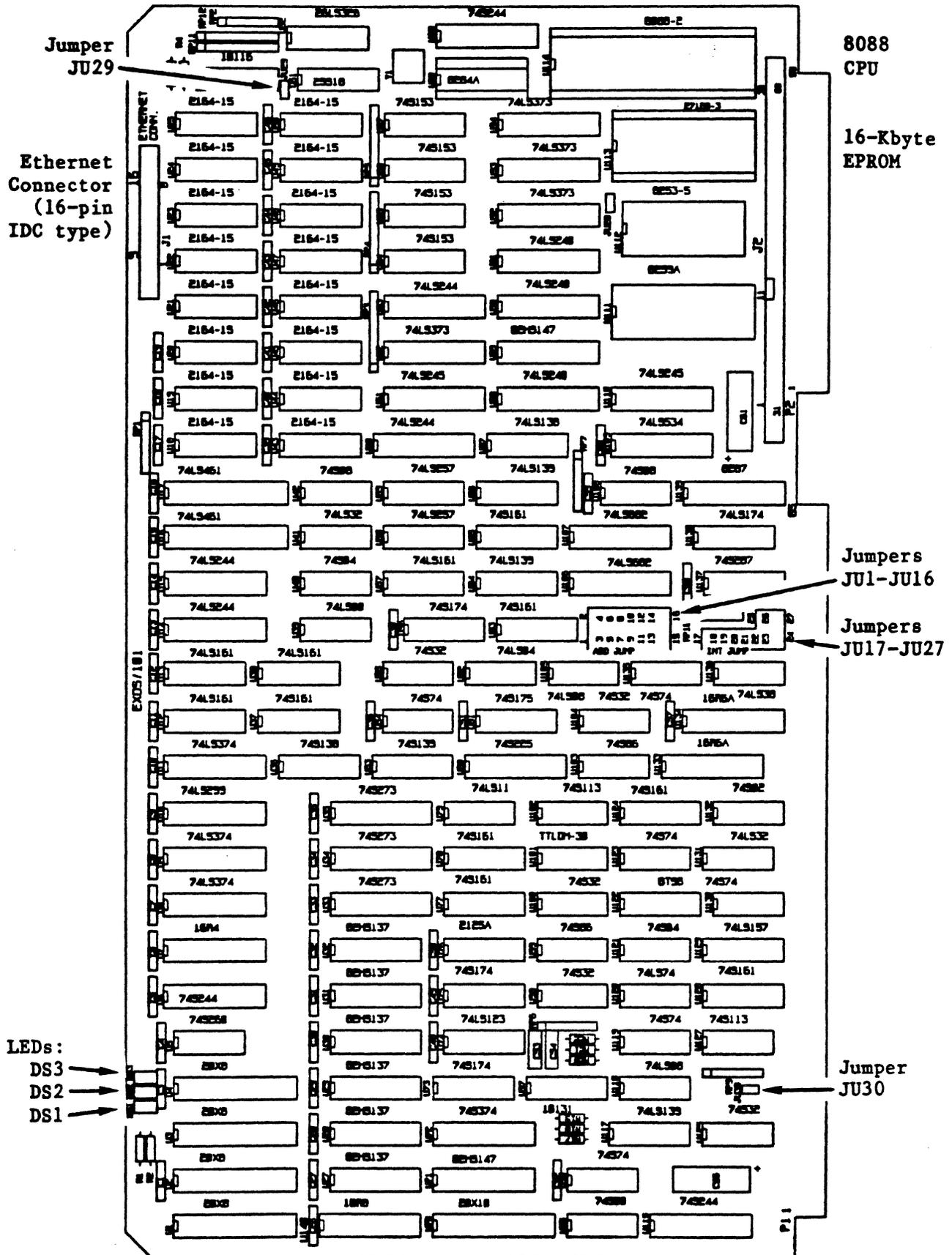
Figure 11-4: Self-Diagnostic and Configuration Error Codes

5.6 A at +5 Volts
 0.5 A at +12 Volts

11.8. Operating Environment

Temperature: 5 to 55 degrees C
 Humidity: 0 to 90% without condensation

EXOS/101 COMPONENT LOCATION



29 November 1983

Attention
Systems Programmers Responsible for Integration
of the
EXOS/101 Ethernet Front-End Processor

Please find enclosed a Document Change Notice pertaining to the EXOS/101 Ethernet Front-End Processor Reference Manual dated August 1, 1983. Please be sure that this information is communicated to all appropriate persons in your organization.

Changes described by the enclosed DCN reflect errors in the most recent manual revision, and do not imply that product functionality has been changed. Any changes listed here will be incorporated in the next revision of the manual. In the mean time, it is suggested that extant copies be marked to reflect these changes, or that the DCN be attached to these manuals.

Note that at least one error corrected by this DCN could cause failure of software written according to the original specification.

If you have any questions about this DCN, or about the EXOS/101 in general, please call me at (408) 945-9526 X230

Sincerely,



George Powers
Marketing Support Engineer

Document Change Notice No0001

DCN Release Date29 November 1983

Document Affected:

Name: Exos/101 Ethernet Front-End Processor Reference
Manual

Date: August 1, 1983

Changes are listed below, according to their order in the
affected document.

1) Figure 4-3

Contained the following, improper C language source
statement:

```
while ((read_port(B) & READY_BIT) == 1);
```

Actually, this statement should be:

```
while (read_port(B) & READY_BIT);
```

Note that the former version of this statement is
effectively an infinite loop.

2) Section 4.4.6 (also affects Figure 4-4):

Stated that the value of the reserved field (at an
offset of six bytes) in the configuration request
message should contain the value 0.

Actually, the first byte of this field should contain
the value 1 (bit 0 is set) in the request message.
The other two bytes should still be set to 0. In the
reply message, all bytes of this field are still
undefined.

Note that improper initialization of this field can
cause undefined (and undesirable) results. In
particular, some (but not all) Revision D, Model B
boards may cease to operate about 20 minutes after
being initialized in front end mode. If a board
flashes the OBAH error code (see Figure 11-4) on its
Status LED, this is the most likely cause.



3) Section 4.4.15

Stated that the default value for the number of hosts parameter in the configuration request message is 1.

Actually, the default value (effective upon first configuration when the value 0FFH is supplied in the request message) is 0.

4) Figure 10-2

Showed a request value of 1080H for the Subtype field in the net boot message header.

Actually, the correct value of this field is 0 (1080H is the proper value for the Ethernet type field in net boot packets).

