EXOS 204

Ethernet Front-End Processor For Unibus Systems

Reference Manual

Publication No. 4200009-00
Revision A March 22, 1985

Excelan Inc.
2180 Fortune Drive
San Jose, CA 95131

NOTICE

This document reflects the features and specifications of the EXOS 204 Ethernet Front-end Processor, and the NX 200 firmware version 4.2. 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 204
NX
NX 200

Ethernet is a trademark of Xerox Corporation.

Unibus is a trademark of Digital Equipment Corporation.

EXOS 204
Ethernet Front-End Processor
For Unibus Systems
Reference Manual

REVISION HISTORY

REVISION	DATE	SUMMARY OF CHANGES
P1	10-25-84	Preliminary-1 Release. EXOS 204 Ethernet Front-End Processor Reference Manual Publication No. 4200009-00
A	03-22-85	Upgrade of Revision P1 to Revision A. Several technical/ editorial changes.

PREFACE

This document describes the EXOS 204 Ethernet Front-End Processor board. It covers information necessary to integrate the EXOS 204 in a Unibus-based system, and to design software both for the host and the EXOS 204. Ethernet and Unibus are described in readily available documents; this manual makes no special effort to explain these standards.

By design, the EXOS 204's operating system kernel (NX 200) 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 200 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.

Section 1 of this manual outlines the principle features of the EXOS 204.

Section 2 is a guide to useful references.

Section 3 describes conventions and restrictions which are crucial to successful application of the EXOS 204.

Section 4 describes initialization of the EXOS 204, including software down-load from the host.

Section 5 discusses using the EXOS 204 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 200 firmware, which provides support for software down-loaded to the EXOS 204. Section 6 describes the real-time, multitasking OS kernel services, and describes the programming environment aboard the EXOS 204. Sections 7 and 8 cover the Ethernet and host interface facilities, which are implemented in NX 200. They are broken out into separate chapters because NX 200's design makes them conceptually detachable.

Section 9 defines the NX 200 kernel calls, and is intended for ready reference once NX 200 services are understood functionally.

Section 10 describes the EXOS 204's network bootstrap protocol, which can be used to automatically down-load software to the EXOS 204 over the Ethernet at initialization time.

Section 11 provides necessary information about EXOS 204 hardware.

(blank page)

TABLE OF CONTENTS

FIGURES

TABLES

(blank page)

1. INTRODUCTION

This section provides an overview of the EXOS 204's features and specifications, and describes its principal modes of operation.

1.1. Overview

The EXOS 204 is a high-performance, front-end communications processor that connects a Unibus system to an Ethernet local area network. It implements the complete Ethernet Data Link Level interface, with significant functional extensions, on a single quad-size Unibus board. In addition, the EXOS 204 can support high-level network protocols on-board, thereby offloading this burden from the host CPU.

The EXOS 204 is directly managed by an on-board 80186 CPU, which runs the NX 200 operating system, stored in two 16-Kbyte EPROMs. A host system controls the EXOS 204 primarily thcugh command and reply messages located in memory accessible from the Unibus. NX 200 firmware interprets the command messages and generates the replies.

NX 200 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 down-load high-level protocols to run on the EXOS 204. Instead, NX 200 firmware brings the EXOS 204's Data Link controller functions out to the host interface. The host system obtains Data Link services through standard request/reply messages; the EXOS 204's RAM is entirely available for buffering packets.

In **front-end processor mode,** the host system down-loads protocol software to the EXOS 204 at initialization time (or the EXOS/204 bootstraps itself from the Ethernet). This software then uses NX 200's real-time, multi-tasking process management services and I/O drivers to control the EXOS 204's Ethernet interface and manage communications with the host system. Standard protocol modules for the EXOS 204, 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 204 under NX 200. This manual contains all information required to write software for the EXOS 204. NX 200 is designed to greatly facilitate this process.

First, NX 200 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 200 is to hide the implementation details of EXOS 204 hardware from user software by providing suitable abstractions for all facilities.

1.2. EXOS 204 Hardware Description

Figure 1-2 shows a block diagram of the EXOS 204. Architecturally, the EXOS 204 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 128 Kbytes of dual-ported RAM.

The EXOS 204 implements the Ethernet Data Link protocol using the 82586 LAN Coprocessor. Functions such as address recognition, CRC check, and buffer chaining

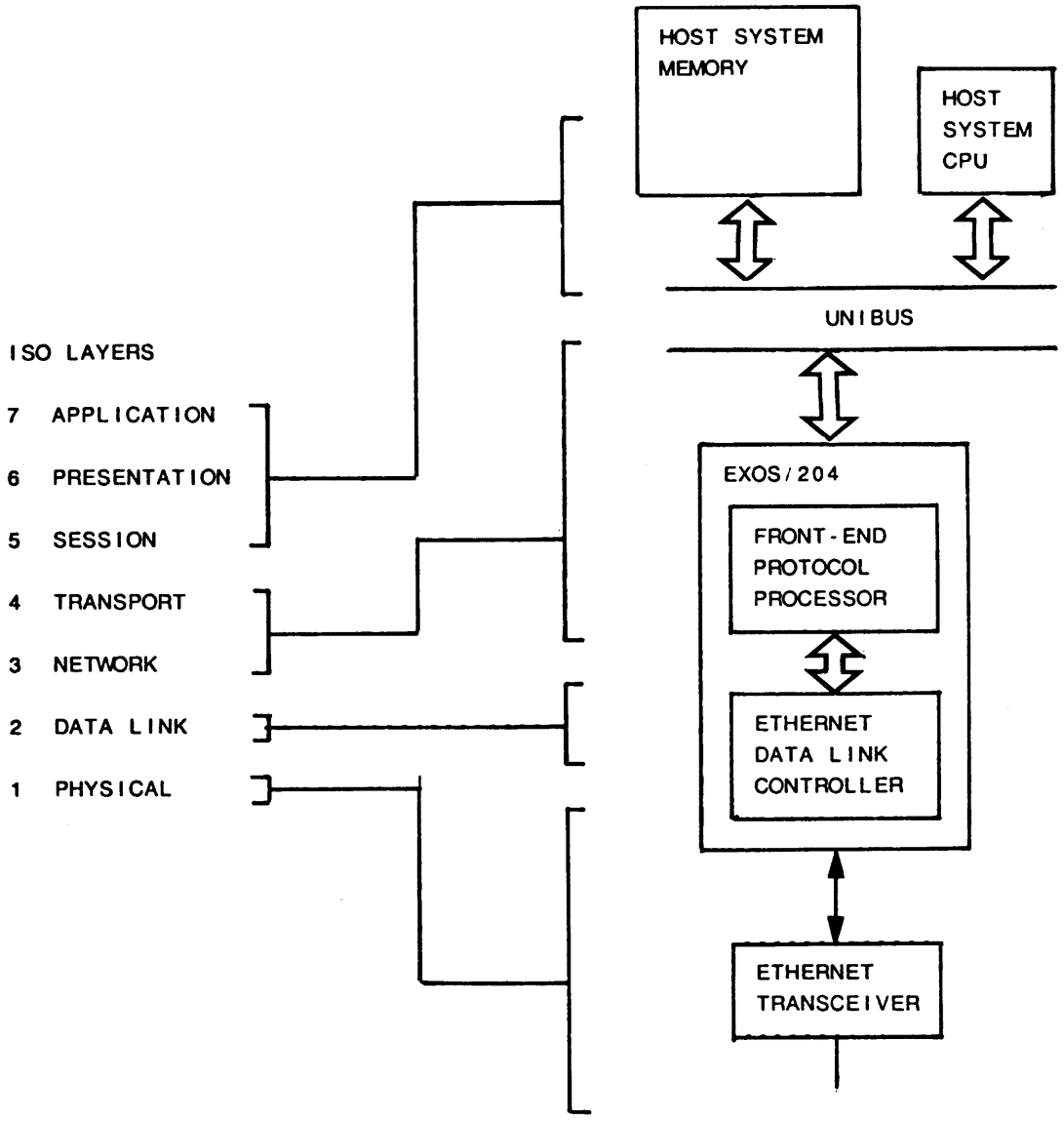


Figure 1-1: An EXOS 204 Front-End Processor Mode Implementation

are managed in hardware, so that the 80186 CPU is fully available for front-end processing applications. The protocol processing engine is supported by either 128K (Model 2) or 256K (Model 3) of RAM. Two 16-Kbyte EPROMs contain Excelan's NX 200 firmware, which includes self-diagnostic tests, an operating system kernel, and network bootstrap code.

1.2.1. Principal Features

- One quad-sized (10.44" by 8.9") Unibus board which occupies one SPC slot.

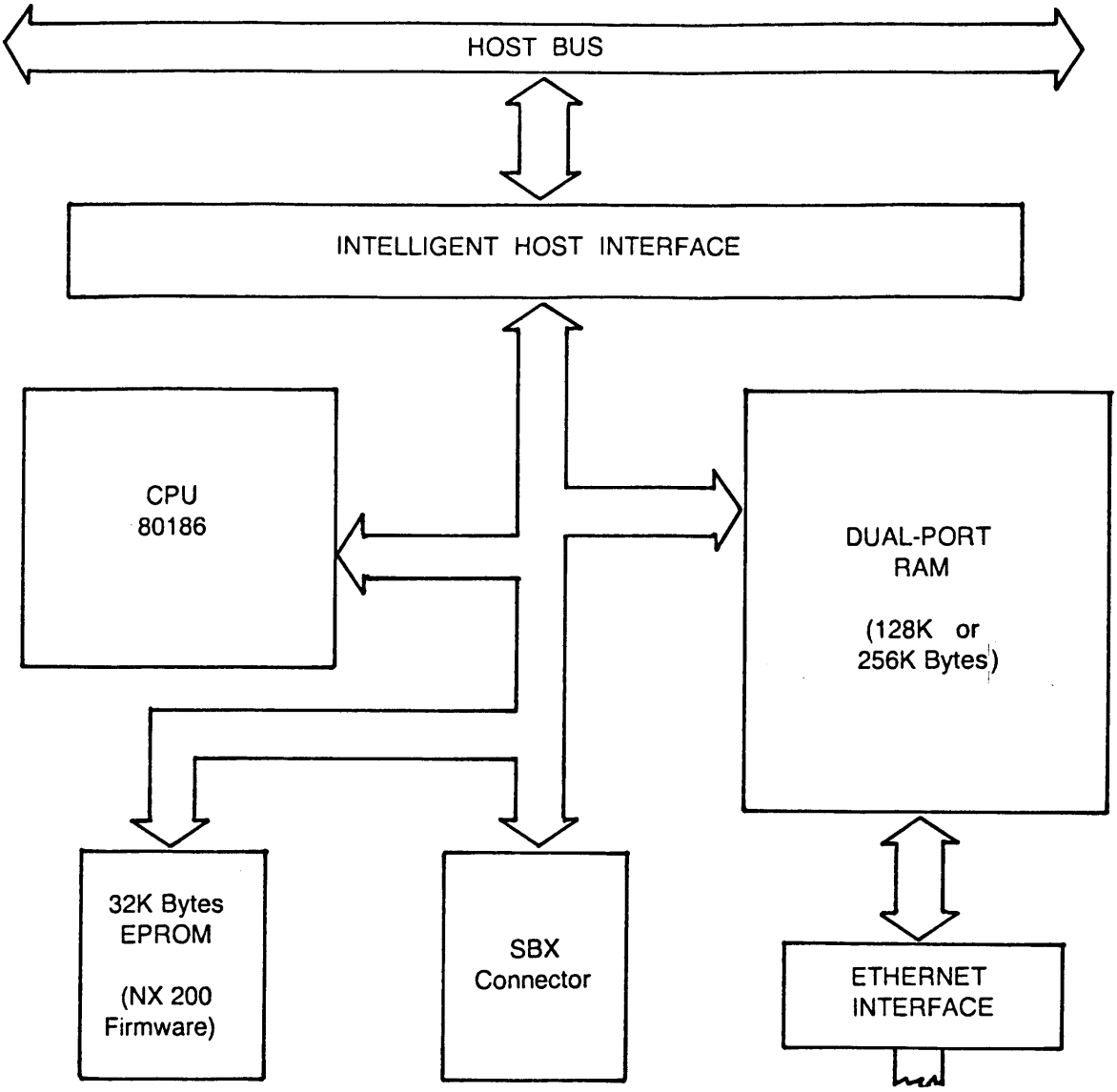


Figure 1-2: EXOS 204 Block Diagram

- On-board 8 MHz 80186 microprocessor (8 MHz clock) and 128 Kbytes of RAM (256 Kbytes on Model 3) support high-level network protocols on-board.
- Dual-ported memory allows concurrent, full-speed access by network hardware and on-board processor.
- 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.
- Hardware recognition of physical, broadcast, and 252 multicast addresses, in addition to promiscuous mode.

- 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.

1.2.2. Ethernet Compatibility

The EXOS 204 conforms fully with Ethernet version 1.0 (September 1980) and version 2.0 (November 1982) specifications published by DEC, Intel and Xerox. Integrated with a standard Ethernet transceiver, it provides all Data Link and physical layer services. The EXOS 204 is also compatible with the IEEE Standard 802.3.

1.2.3. Unibus Compatibility

The EXOS 204 conforms with Unibus specifications by DEC as a 16-bit master. Compliance is 8-bit and/or 16-bit transfers, 18-bit addressing, and bus-vectored interrupts.

1.2.4. Unibus Interface

The EXOS 204 can access the entire Unibus system memory space (256 Kbytes)including the full 8-Kbyte I/O space, as a 16-bit bus master. One-byte communication path is provided from the Unibus to the EXOS processor via an I/O port. This is used during initialization to transmit the address of a communication area in the shared Unibus memory.

The EXOS 204 and host processors can interrupt each other. The board generates bus-vectored interrupts to interrupt the host. Interrupt priority can be set from level 4 to level 7, via jumper selection. The host can interrupt the EXOS 204 processor by writing to an I/O port.

1.2.5. Ethernet Functions

The EXOS 204 performs all physical and Link Level Ethernet functions except for transceiver functions. These include:

- serial/parallel and parallel/serial conversion.
- address recognition.
- framing and unframing of messages.
- Manchester encoding and decoding.
- preamble generation and removal.
- carrier sense and deference.
- collision detection and enforcement, including jamming, backoff timing and retry.
- FCS (CRC) generation and verification.
- 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 204 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 204 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:

- 8 MHz 80186 CPU, with on-chip clock timer and interrupt controller, operating at 8 MHz.
- 128K dual-ported RAM (plus 128K additional dual-ported RAM on Model 3).
- 32 Kbytes of EPROM, containing NX 200 firmware.

Firmware supplied with the board (the NX 200 Network Executive) provides simplified Ethernet and host interface device drivers, and a multi-tasking environment for high-level network protocols.

1.3. NX 200 Firmware Description

NX 200 resides in EPROM memory, which appears at the high end of the 1M byte address space of the 80186. NX 200 data structures use 4K of the RAM space; the rest is available for higher level software. Figure 1-3 provides a graphic representation of the NX 200 software architecture.

1.3.1. Principal Features

- Self-diagnostics for testing the integrity of EXOS 204 hardware.
- Booting process that allows higher level software to be down-loaded either from the host or from the network.
- 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.
- Device drivers for the Ethernet controller and host computer interface. Access through message queues simplifies pipelined communications.
- Supports network management functions by collecting network statistics.
- Allows the EXOS 204 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 200 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 204 either boots itself from the Ethernet, or awaits initialization by the host system, depending on a jumper option on the board.

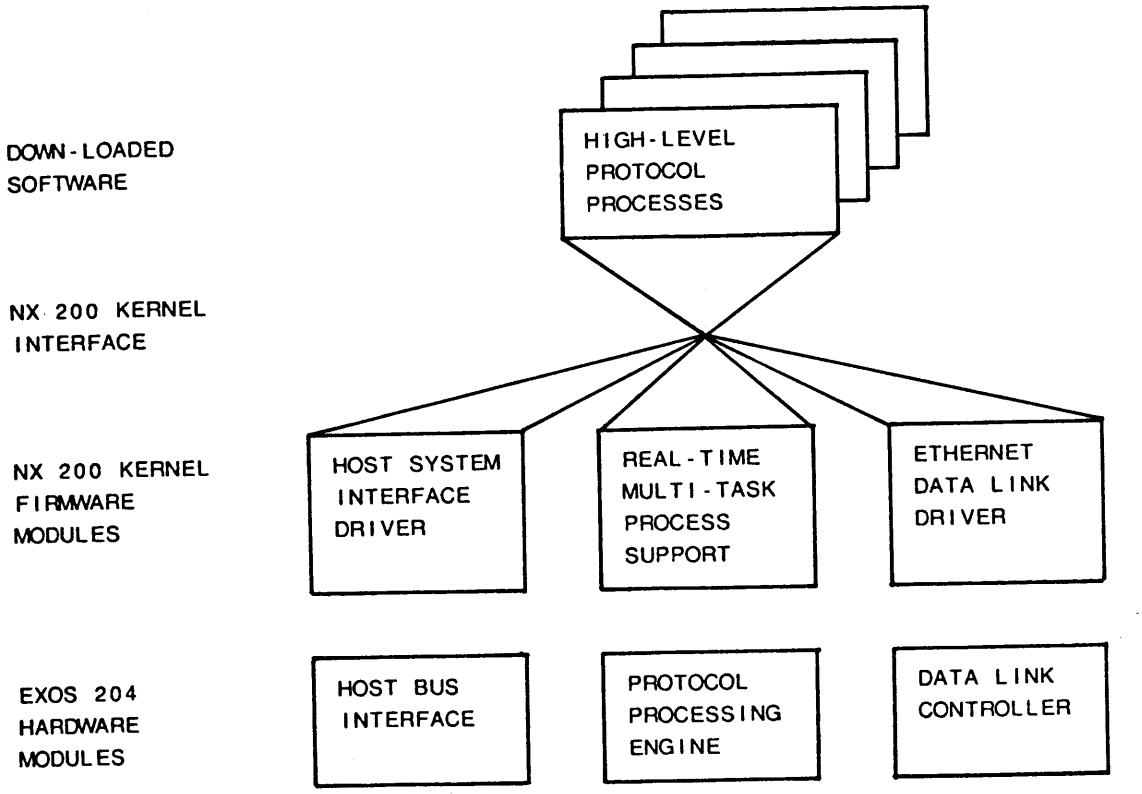


Figure 1-3: NX 200 Software Architecture

If the jumper selects initialization by a host system, the host then uses a configuration message to select NX 200's mode of operation, and specify several other parameters. It can down-load software directly, tell NX 200 to boot itself from the Ethernet, or select link level controller mode. If initialization includes down-loading software, then NX 200 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 204.

1.3.3. Multi-tasking support

NX 200 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 200's reserved memory there are no restrictions on how memory is used. Processes access NX 200 functions by executing the 80186'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 200 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 200 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 200'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 200 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 204.

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 200 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 204 process.

Apart from transferring data by means of messages, processes on the EXOS 204 can also directly read and write the the host memory by means of NX 200 calls. The contents of messages sent and received from the host is not interpreted by the NX 200, 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 204 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 200 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 200.

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 204 by keeping a tally of various events such as the number of packets transmitted/received, packet errors etc.

1.3.7. Ethernet Link Level Controller Mode

If the EXOS 204 is to be used in link level controller mode, then most of the description above of NX 200 can be disregarded. In this mode, the host does not down-load 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 204 returns a reply message. Transmit and receive commands can be pipelined -- NX 200 uses 60 Kbytes of the dual-ported RAM for buffering packets.

2. REFERENCES

The EXOS 204 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).

[2] DEC, Intel, and Xerox Corporations, "The Ethernet: A Local Area Network: Data Link Layer and Physical Layer Specifications." Version 2.0 (November, 1982)

The EXOS 204 conforms to the Unibus specifications:

[3] **Digital Equipment Corporation, PDP-11 Architecture Handbook , Order Code: EB-23657-18 (1983).**

The EXOS 204 uses the 82586 LAN Coprocessor for implementation of Ethernet Data Link protocol:

[4] Intel Corp., **LAN Components User's Manual, Document No. 230814-001, Intel Corp., (1984).**

The EXOS 204 supports front-end processing of user-written higher-level protocols, on an 80186 CPU:

[5] Intel Corp., **iAPX 86/88, 186/188 User's Manual, Document No. 210911-001, Intel Corp., (1983).**

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

[6] 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:

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

(blank page)

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 204 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 204 require the consideration of two different programming environments: that of software on the EXOS 204 itself, and that of software on a host computer which communicates with the EXOS 204. In either environment, it is crucial that user software store data objects which are known to NX 200 firmware in the order which NX 200 expects - and that the programmer understand how NX 200 stores data objects which are known to user software.

In the EXOS 204's own memory address space, NX 200 always interprets data in the 80186 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 204 memory address 1C83H would appear as follows:

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

In the Unibus memory address space shared between the EXOS 204 and the host system, NX 200 can interpret data either in the 80186 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 200 interprets must appear to the EXOS 204 in the 80186 CPU's native order.

If the conversion option is enabled, then NX 200 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 200 must know the data type to apply the appropriate conversion, the word and longword conversion are applied only to data objects which NX 200 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 204 memory.

3.7. Data Alignment

The EXOS 204 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 200 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 204 does not generally require even alignment, this practice is still recommended, in order to obtain the optimum performance.

3.8. 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 200'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 200 depends on context. Any address which refers to a location in EXOS 204 memory, whether the address value itself is stored in EXOS 204 memory or in host memory, is always interpreted as a **segmented address.** This term refers to the 80186 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 80186 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.

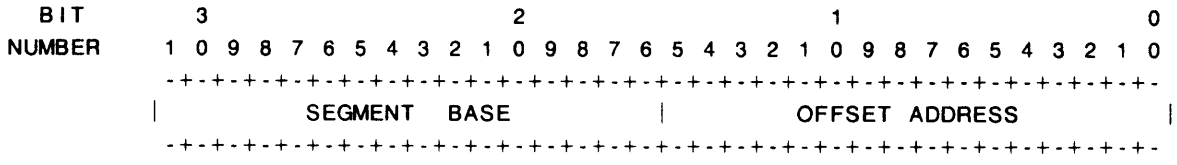


Figure 3-1: Mapping of Segmented Address into Longword Data Type

When a segmented address is stored in EXOS 204 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 204 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 204's **host address mode option.** In segmented mode, they are interpreted in the same manner as addresses referring to EXOS 204 memory locations. The NX 200 also provides an absolute host address mode. An absolute address is a

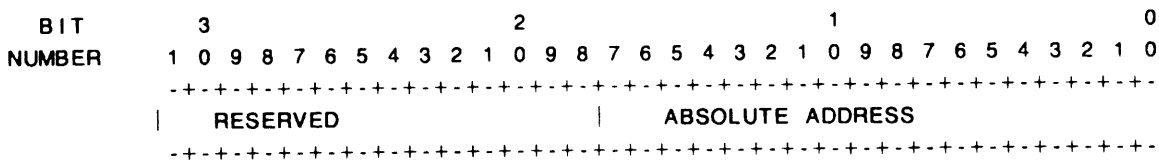


Figure 3-2: Mapping of Absolute Address into Longword Data Type

simple 18-bit physical memory address, mapped into the longword data type as shown in Figure 3-2.

As shown in the figure, the most significant 14 bits are reserved, and should be set to 0. When an absolute address is stored in EXOS 204 memory, it appears in the following order:

Byte 0	(bits 0-7):	least significant byte
Byte 1	(bits 8-15):	somewhat significant byte
Byte 2	(bits 16-23):	most significant byte (low-order 2 bits only); the high-order 6 bits must be 0
Byte 3	(bits 24-31):	reserved; must be 0

Storage order in the host system memory should appear the same to the EXOS 204 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.

3.9. Shared Unibus Memory Access Restrictions

It is the user's responsibility to ensure that a specified Unibus memory address exists in functional memory. If an invalid address is specified and the EXOS 204 attempts to access it, then, depending on whether or not the "time out" option has been jumper-selected or not, one of the two things can happen:

If the time out option is not selected, then the EXOS 204 does not time out if no memory response is received on the Unibus. To aid diagnostics, four Unibus Status LEDs are provided. Their location on the EXOS 204 is shown in Section 11. When the LED DS4 is lit, the EXOS 204 is accessing the Unibus. Thus if the LED is constantly lit then most likely the EXOS 204 has been given a non-existent address and is stuck waiting for the response.

If the time out option is selected, then after 16 microseconds the EXOS 204 goes offline and the status LED flashes the error code BA (Hex) at regular intervals.

The EXOS 204 can access data structures anywhere in the 256 Kbyte Unibus memory space. It accesses this address space by dynamically mapping two consecutive 64-Kbyte windows of its own CPU's address space into Unibus memory. User software does not perform either the mapping or the data transfer; it simply gives addresses to NX 200 firmware, which effects the transfer. Data structures that straddle beyond the 64-Kbyte bound are automatically accessed via the second window without any remapping.

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 204 when it is installed in a Unibus-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 204, 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 200 firmware, uses these queues for all further communications between the host processor and the EXOS 204.

The following paragraphs give an overview of the initialization process:

1) The host system resets the EXOS 204, which then executes self-diagnostics. If the diagnostics fail, then the EXOS 204 displays an error code on the NX 200 status LED (see Section 11) until reset again. If the diagnostics pass, then the EXOS 204 awaits configuration by the host.

2) The host system passes the EXOS 204 the address of a configuration message in host memory. The EXOS 204 examines this message, and modifies some fields according to the results of configuration. If configuration is unsuccessful, then the EXOS 204 again displays an error code on the NX 200 status LED until reset. If the configuration message is valid, then the EXOS 204 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 204 begins to execute firmware which brings NX 200'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 204, 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 204 proceeds to bootstrap itself from the Ethernet interface, as described in Section 10. Depending on how the bootstrap server configures the EXOS 204, 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 204 is conducted via a coordinated exchange of interrupts, I/O instructions, and data transfers through shared memory on the Unibus. 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 204

The host's means of active access to the EXOS 204 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 Unibus, and can be both read and written. Their addresses are selected by jumpers on the EXOS 204, described in Section 11.

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

Read A: No Operation.

Write A: resets the EXOS 204 (see Section 4.3).

Read B: returns the EXOS 204 status byte:

- Bit 0: (Error Bit) when 0, indicates a fatal error in EXOS 204. When the EXOS 204 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 204 has failed the self diagnostics.
- Bits 1-2: undefined.
- Bit 3: (Ready Bit) when 0, indicates that the EXOS 204 is ready to accept a byte written into port B. When 1, the EXOS 204 has not yet read the byte last written into port B.
- Bits 4-7: undefined.

Write B: interrupts the EXOS 204 CPU, and communicates a 1-byte value. This is the only way to communicate a value to the EXOS 204 other than through shared memory.

4.1.2. EXOS 204 Access to the Host

The EXOS 204 functions as a master on a Unibus system. It can access the full 256-Kbyte memory address space which includes the 8K I/O address space, and interrupt the host processor. User software on the EXOS 204 does not directly control these resources. Instead, it calls NX 200's host interface driver, described in Section 8.

In general, data is transferred between the host and the EXOS 204 via shared memory, which may be any portion of system memory accessible to both processors on the Unibus. The EXOS 204's 80186 CPU performs the transfer by dynamically mapping part of its own address space into the Unibus memory address space, and executing a block transfer instruction or, under some circumstances, using the DMA on the 80186. Note that the EXOS 204's on-board memory cannot be shared; it is not directly accessible by the host processor.

The EXOS 204 can interrupt the host either through memory addresses or the Unibus interrupt lines. The type which will be used is selected at initialization time. Memory interrupt addresses are configured by software; the interrupt line is selectable by means of a jumper option, described in Section 11.

4.2. Host Data Order Conversion Option

The host data order conversion option determines whether the EXOS 204 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 200 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 200 will then apply the appropriate conversion to all data objects subsequently read from host memory.

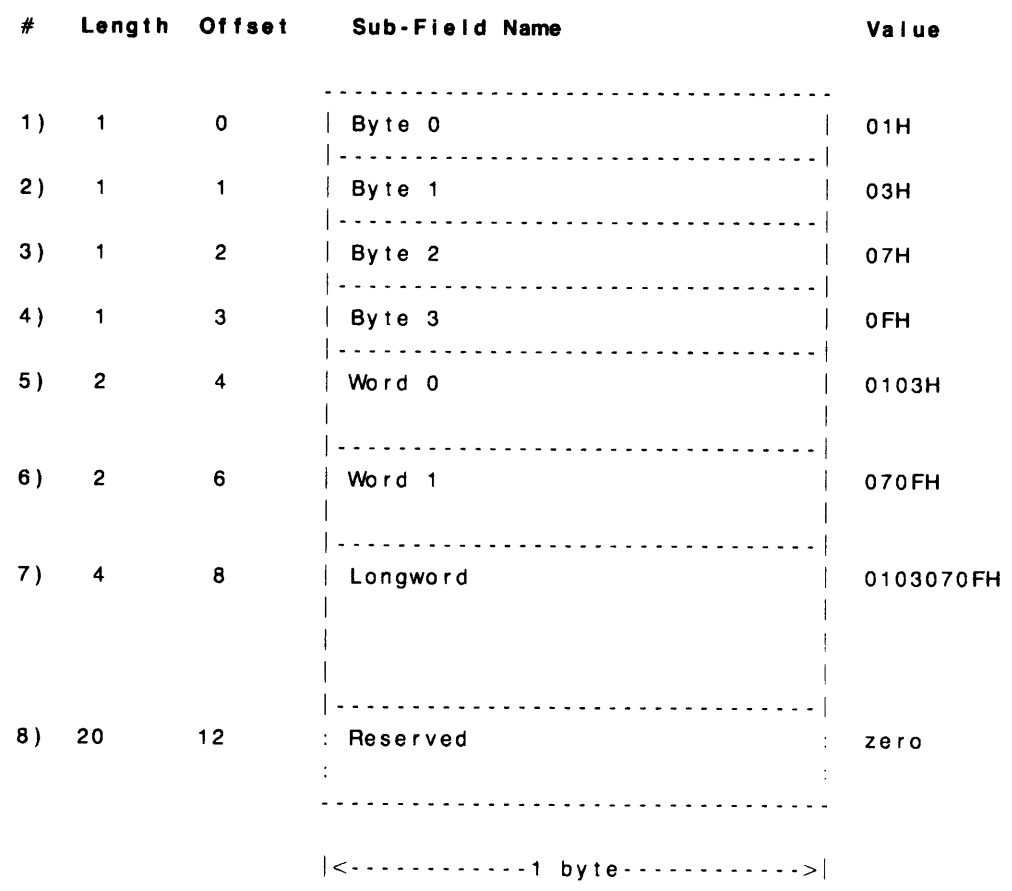


Figure 4-1: Host Data Order Conversion Option Test Pattern

For the word data type, NX 200 can swap bytes if necessary. For the longword data type, NX 200 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 204.

Naturally, the EXOS 204 must know the type of a data object to apply the appropriate conversion. All data objects described in this section are known to NX 200, except for the actual contents of messages between the host and the EXOS 204. NX 200 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 204. For instance, the firmware which drives the Link Level Controller Mode (see Section 5) runs at user level under NX 200, and converts word and longword data objects which are known to itself, but not to NX 200. NX 200 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.

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 200 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.

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.

The EXOS 204 is reset by the Unibus INIT signal, or whenever port A is written from the Unibus. Host software should use the latter method to be sure. On reset the EXOS 204 performs a series of self tests to confirm hardware integrity. While these tests run, the NX 200 status LED (see Section 11) will remain lit constantly. When self-diagnostics complete successfully, the EXOS 204 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 204 repeatedly reports an error code through the NX 200 status LED (for error code values, see Section 11). The EXOS 204 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 204 will attempt to down-load software over the Ethernet (see Section 10). Otherwise the EXOS 204 awaits configuration by the host processor.

```
/* 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 rsrvd[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.rsrvd[i] = 0;
}
```

Figure 4-2: Host Data Format Test Pattern Initialization

The host configures the EXOS 204 by passing it the address of a configuration message, located in shared memory. This message establishes various NX 200 parameters and selects among several modes of operation. Parameters include memory allocation for NX 200 objects, the address of NX 200's movable data area in EXOS 204 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 200 will first note the contents of the host configuration message. Other configuration options include host data order conversion and the host address mode.

```
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) >;
    write-port(A);  /* reset the EXOS 204 */
    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);
        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

The host processor communicates the address of the configuration message to the EXOS 204 by writing a sequence of 8 bytes into port B. Each byte should be written after checking that the ready bit of the EXOS 204's port B is clear. This ensures that the EXOS/204 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** Unibus memory address (least significant byte first). The configuration message must be aligned on a even address boundary. When the last byte is written, the EXOS 204 reads and interprets the configuration message. If the address for the initialization message is not valid, then the EXOS 204 will display an error code on the status LED (see Section 11).

When the EXOS 204 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 200 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 between the host and the EXOS 204 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 200. 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 200 and the EXOS 204 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 0FFH in the request message. The EXOS 204 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 200 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

#	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	0FFH	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	EXOS Context	zero	see text
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)

host has given the EXOS 204 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 204 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.

ACH invalid number of hosts.

ADH invalid host message queue parameter. NX 200 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 204 Operation Mode Field

The EXOS 204 operation mode field determines the mode in which the EXOS 204 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 204 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 204 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 204 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 204 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 200 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 200 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 200 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. EXOS Context

This 3-byte field returns the EXOS context information. In the request message the value of this field must be zero. In the reply message, the middle byte (offset 11) returns the context value; the other two bytes are undefined. For the EXOS 204 the context value must be 04.

4.4.7. Host Address Mode Field

The host address mode field determines how NX 200 will interpret addresses which refer to objects in host memory. It is defined bitwise:

Bit 0: Set Mode Bit. If 0, NX 200 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 200 will interpret all addresses as 80186-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 204 memory for user software. This field contains a valid value only if the EXOS 204 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 204. 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 204 is configured in mode 1 or mode 2. If the optionl 128K of RAM between 20000H and 3FFFFH is either absent or is malfunctioning, then the map will not contain this segment.

4.4.11. NX 200 Movable Block Address Field

The NX 200 movable block address field can be used to redefine the location of NX 200's movable data area, described in Section 6.2. If the EXOS 204 is configured in mode 0, this field must be 0FFFFH, 0FFFFH. In modes 1 or 2, the value 0FFFFH, 0FFFFH 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 0FFFFH.

In the reply message, this field returns the actual address of the NX 200 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 200 will support. If the EXOS 204 is configured in mode 0, this field must be 0FFH. In modes 1 or 2, the value 0FFH 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 200 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 200 will support. Note that this number does not include system mailboxes. If the EXOS 204 is configured in mode 0, this field must be 0FFH. In modes 1 or 2, the value 0FFH 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 200 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 200 will support. Note that this number does not include the physical, broadcast, universal, or null slots, which are permanently allocated. The value 0FFH 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 200 will support.

4.4.15. Number of Hosts Field

The number of hosts field specifies the number of host CPUs on the Unibus interface. Permissible values depend on the mode of operation. In all modes, the value 0FFH 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 204 (see Section 4.5). Addresses for all message queue data structures are 16-bit offsets, calculated relative to this base. NX 200'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.

In absolute mode this field contains a 18-bit absolute memory address, also stored as a longword. The lower-order 18 bits contain the address; the remaining high-order 14 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 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 204 will use to alert the host of a change in the status of the Host-to-EXOS 204 message queue. Options are:

0 no interrupt. The host polls the message queues.

1 undefined.

2 memory mapped. The EXOS 204 writes a specified value at the specified memory address.

3 undefined.

4 bus-vectored interrupt.

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 memory mapped interrupt type. If interrupt type 2 is selected, then this value will be written to the specified 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 memory mapped and bus-vectored interrupt type. If interrupt type 2 is selected, then it contains a Unibus memory address, which NX 200 will interpret according to the host address

mode. If interrupt type 4 is selected, then the first word contains an interrupt vector; contents of the second word are undefined. 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 204 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 204 will use to alert the host of a change in the status of the EXOS 204-to-host message queue. Options are:

0 no interrupt. The host polls the message queues.

1 undefined.

2 memory mapped. The EXOS 204 writes a specified value at the specified memory address.

3 undefined.

4 bus-vectored interrupts.

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 memory mapped interrupt type. If interrupt type 2 is selected, then this value will be written to the specified 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 memory mapped and bus-vectored interrupt types. If interrupt type 2 is selected, then it contains a Unibus memory address, which NX 200 will interpret according to the host address mode. If interrupt type 4 is selected, then the first word contains an interrupt vector; contents of the second word are undefined. The value of this field is preserved in the reply message.

4.5. Message Queue Format

Once the EXOS 204 is configured, message queues in shared memory serve all further communications with the host. This includes software down-load, link level controller mode service requests, and communication with down-loaded protocol code. Two

message queues are maintained by the NX 200 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 204; it reads and modifies its value during message exchange. The host may read it, but must not modify it. The EXOS 204 queue header and all message buffers must lie within a single 64K area of memory, called the queue segment.

Message queue data structures are described here as viewed by NX 200. The configuration message provides NX 200 with the queue segment base and the offset address of the queue header, for each queue. NX 200 regards the queue header value and link field values as 16-bit offsets calculated relative to the queue segment base. As long as this view is preserved for NX 200, users are perfectly free to augment these data structures in any manner necessary to implement the desired mechanisms for the host message handling software.

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

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, and set to 0 in Host-to-EXOS messages. Its value in reply message is undefined.

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 204. The host may alter a message buffer only while it has ownership.

Bit 1: Done bit. The EXOS 204 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 204 and buffers it has already processed.

Bit 2: Overflow Bit. The EXOS 204 sets this bit to 1 if an EXOS-to-Host message had to be truncated because the host buffer's Data Field was shorter than the message sent.

Bits 3-7: undefined. These bits are reserved for the EXOS 204, and should not be used for any purpose by the host.

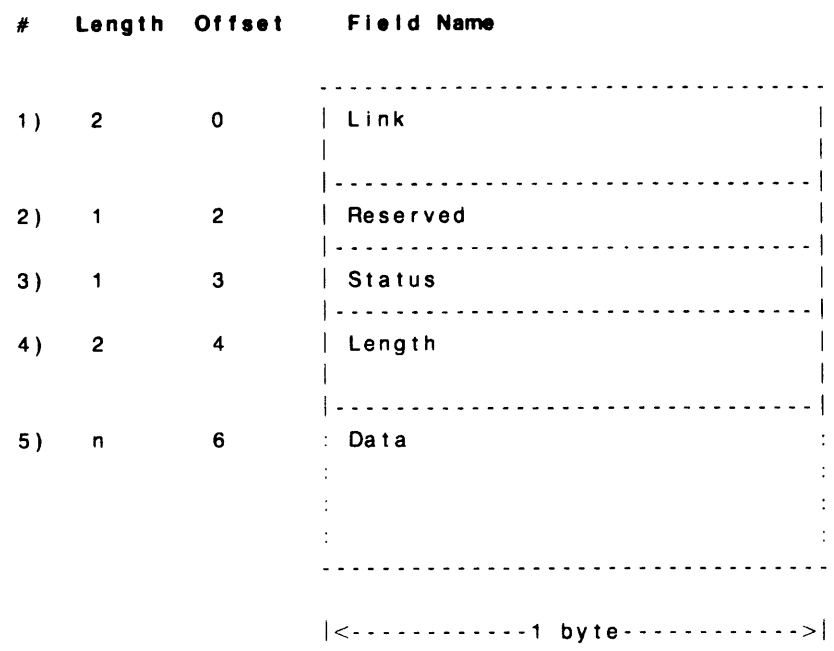


Figure 4-5: Message Buffer Format

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 204. 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.

In Host-to-EXOS messages, set this field's value before passing the message to the EXOS. In EXOS-to-Host messages, this field tells the host how many valid bytes were written into the data field. The host must reset its value to the data field's size before returning a buffer to the EXOS.

4.5.5. Data Field

The data field contains the actual message data passed between the host and the EXOS 204. NX 200 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 204 (see Section 8).

4.6. Message Queue Initialization

The host must initialize the message queues and the queue headers prior to configuring the EXOS 204. 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 204 queue headers should point to the same buffer.

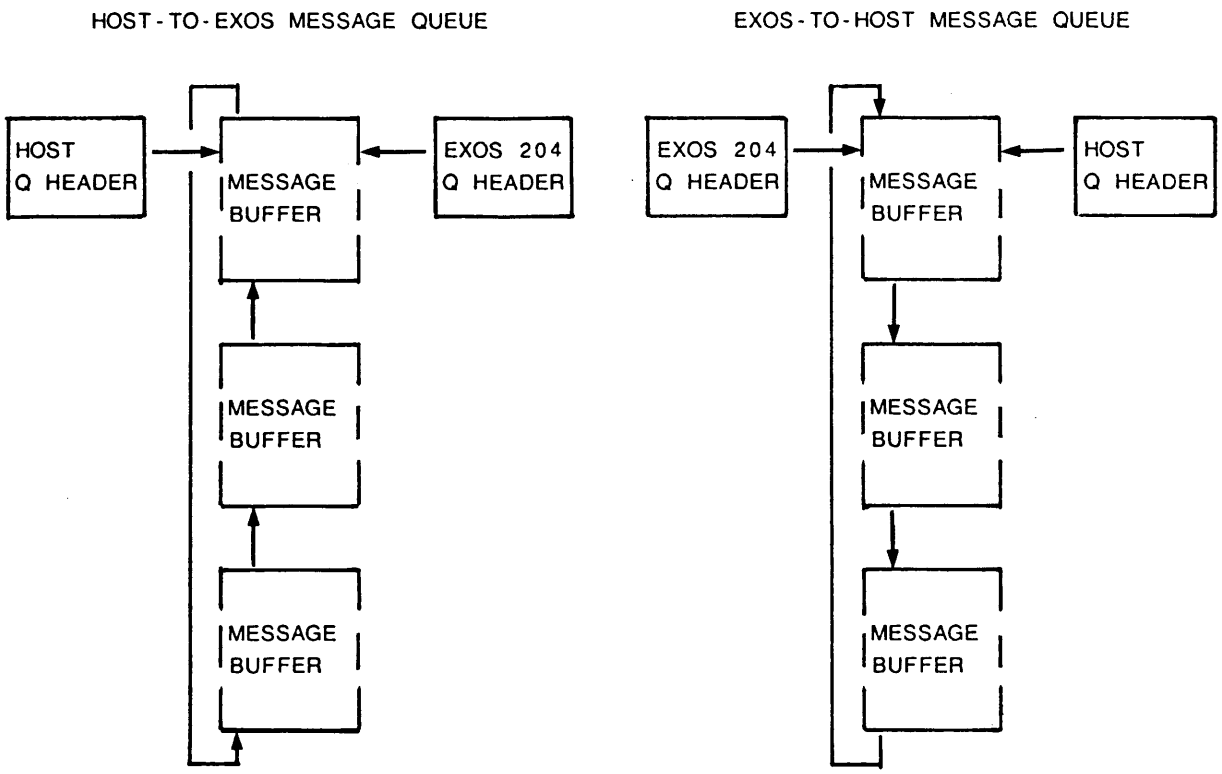


Figure 4-6: Message Queue Data Structures at Initialization Time

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 204. 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 204, 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 a PDP-11 host system, where the EXOS 204 is configured in the segmented host address mode. The configuration message describing the queue is also shown in part. Data structures are shown as vectors containing hexadecimal byte values. The Unibus 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

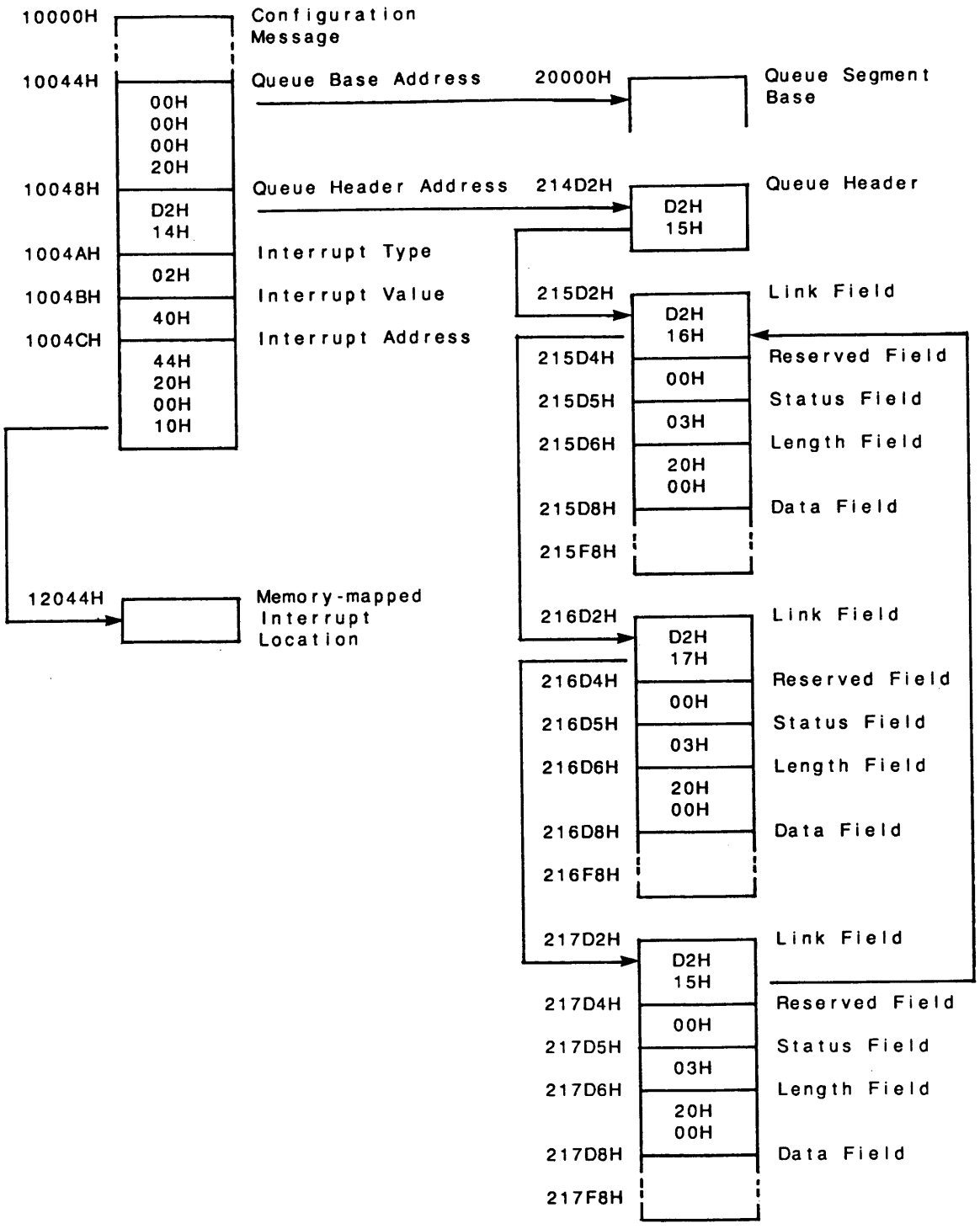


Figure 4-7: Example EXOS-to-Host Message Queue, at Initialization

12044H will interrupt the host. NX 200 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 200 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 200's behavior.

In a typical implementation, the host system and the EXOS 204 each maintain private queue headers for both queues (see Figure 4-6). The EXOS 204's host-to-EXOS message queue's header points to the message buffer which NX 200 will receive next. The EXOS 204's EXOS-to-host message queue's header points to the message buffer which NX 200 will send to next. NX 200 maintains these queue headers after configuration. Although the EXOS 204 queue headers are kept in host memory, after initialization the host should not refer to these. Similarly, the EXOS 204 will not refer to the host's own queue headers. Host queue headers may be of any format (16-bit offset, 32-bit virtual address, array index,etc.) which is most convenient to the host software.

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 204's queue header will always point to the next buffer in which the EXOS 204 will look for a message. Both pointers will always move sequentially through the message queue. Note that unless a message arrives on the next buffer, the EXOS will not scan any further in the queue. This means that the host should always write the message in the next buffer where the EXOS expects it to be rather than in any arbitrary position in the queue. During the course of message processing, the host's queue header may end up several buffers ahead of the EXOS 204's queue header, but should never "lap" it from behind. Any difference between the headers represents buffers which the EXOS 204 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 204's queue header will always point to the next buffer in which the EXOS 204 will send a message. As above, both pointers will always move sequentially through the message queue. Note that unless the next buffer is available to the EXOS 204, it will not scan any further to find a free buffer to write the message. This means that the EXOS will always write the message in the next buffer where the host expects it to be rather than in any arbitrary position in the queue. During the course of message processing, the EXOS 204'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 transfer messages to the EXOS 204 using the following steps:

1) Test the owner bit of the buffer to which the host queue header points. If the EXOS 204 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) Advance the host 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) Set the current buffer's owner bit, so that the buffer now belongs to the EXOS 204.

5) Interrupt the EXOS 204 by writing to port B, to notify it that a new message is available.

The EXOS 204 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 204 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 204 receives an interrupt from the host, it will:

1) Examine the owner bit of the buffer to which its own queue header points. If the buffer belongs to the EXOS 204, 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 its 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 200's host interface mailbox), then wait.

4) Reset 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 204 has a message to transfer to the host, NX 200 will:

1) Test the owner bit of the buffer to which its queue header points. If the buffer belongs to the EXOS 204, 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 200 can process other jobs in the mean time).

2) Load the link field of the current buffer into its 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 to the length actually transferred (it will not exceed data field

length). If the data field was too short for the message, then it sets the overflow bit.

4) Reset 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 204, 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 204 is returning a host-to-EXOS message buffer, or could be spurious.)

2) Advance the host's own queue header, so that it now points to the next buffer in the queue.

3) Extract the message from the current buffer. It may check the overflow bit to be certain that the entire message was sent. If there is no consumer for this data, then wait.

4) Set the length field to the size of the data field.

5) Set the current buffer's owner bit, so that the buffer is returned to the EXOS 204.

6) Interrupt the EXOS 204 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.

While the host is processing an interrupt, the EXOS 204 may in the meantime write more messages into the queue. The host may elect to process these messages in addition to the message associated with the interrupt being serviced. Note, however, that at least one interrupt will remain pending, so that when interrupts are re-enabled, the host will be again interrupted by the EXOS 204, although the corresponding message would have already been processed.

Although the above description assumes that the EXOS 204 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 204 is configured in mode 1, host software would then down-load and run higher level protocol software. Two message formats are provided for this purpose, one to copy user code and data to the EXOS 204, and another to start code execution. For each message the EXOS 204 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 204 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 200 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 200, and must be set to 0. Its value in the reply message is undefined.

4.8.1.2. User Id Code Field

The user id code field is not interpreted by the EXOS 204, 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.

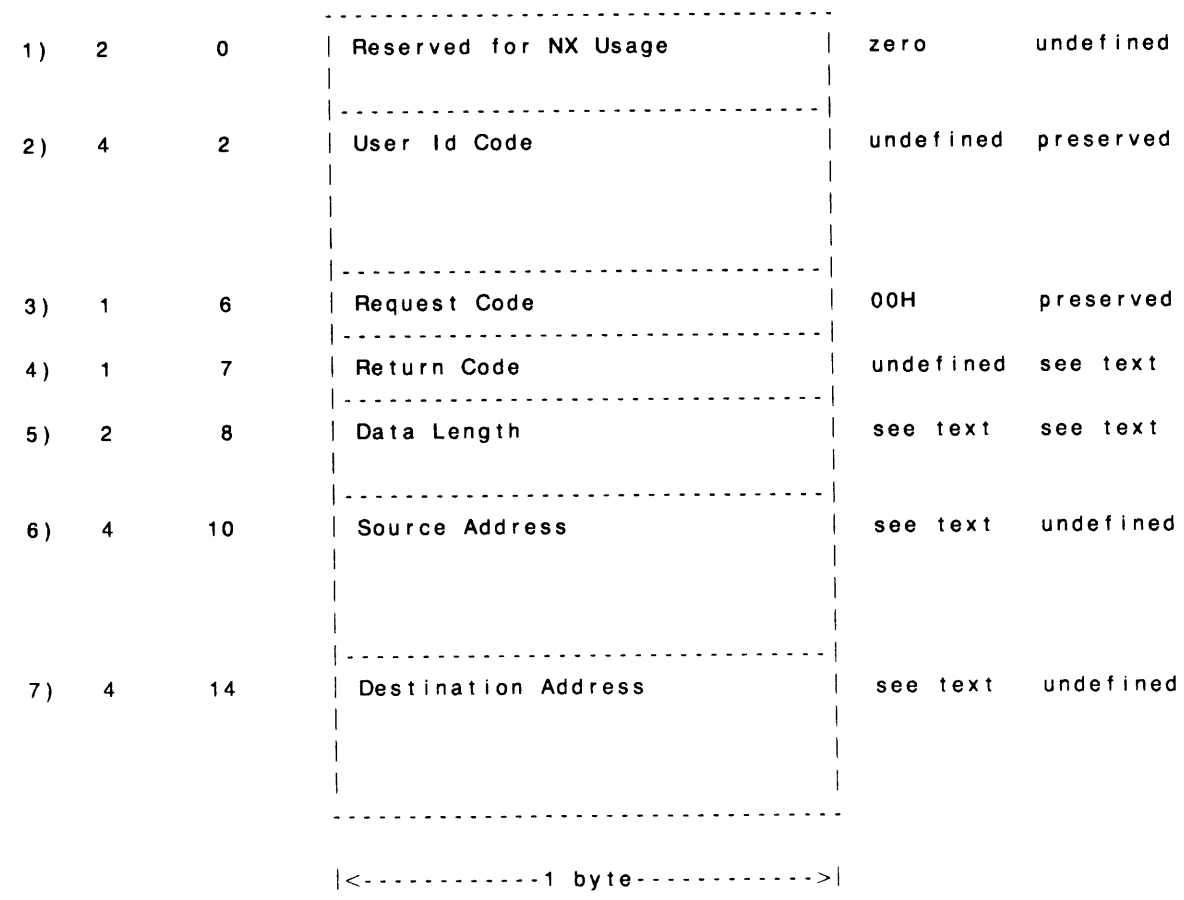


Figure 4-8: EXOS 204 Down-Load Request/Reply Message

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 200, no copy done.
A1H	invalid request, the EXOS 204 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 204 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 204 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

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 204 does not itself process any more messages. Instead, all messages sent to the EXOS 204 will be queued up for user processes running under the NX 200 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 200 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 200, and must be initialized as 0. Its value in the reply message is undefined.

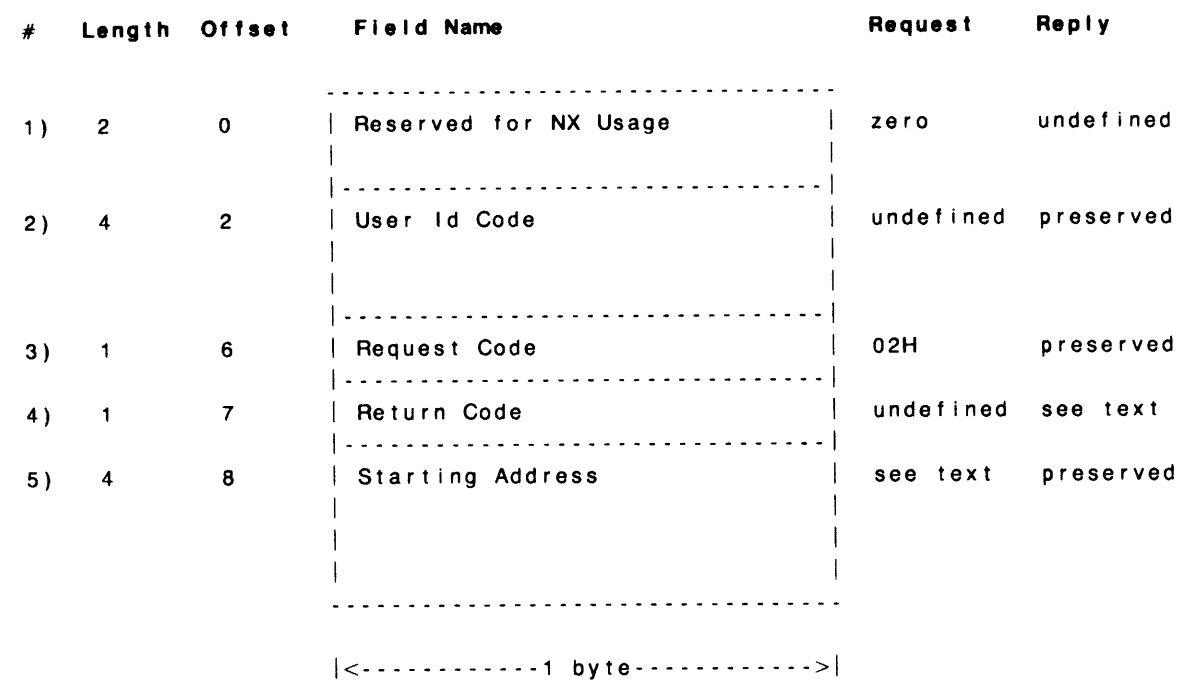


Figure 4-9: EXOS 204 Start-Execution Request/Reply Message

4.8.2.2. User Id Code Field

The user id code field is not interpreted by the EXOS 204, 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 204 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 204 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 down-load software; instead the EXOS 204 runs firmware which brings NX 200'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 204 via the message protocol described in Section 4.5. The EXOS 204 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 200 Ethernet interface for EXOS 204-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 204 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 204 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 204 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_ADDRS	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 200 (see Sections 7.1 and 7.2). The latter four requests correspond exactly to the NX 200 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 204. 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 204. When interrupted, the EXOS 204 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 204 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 204. This should 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.

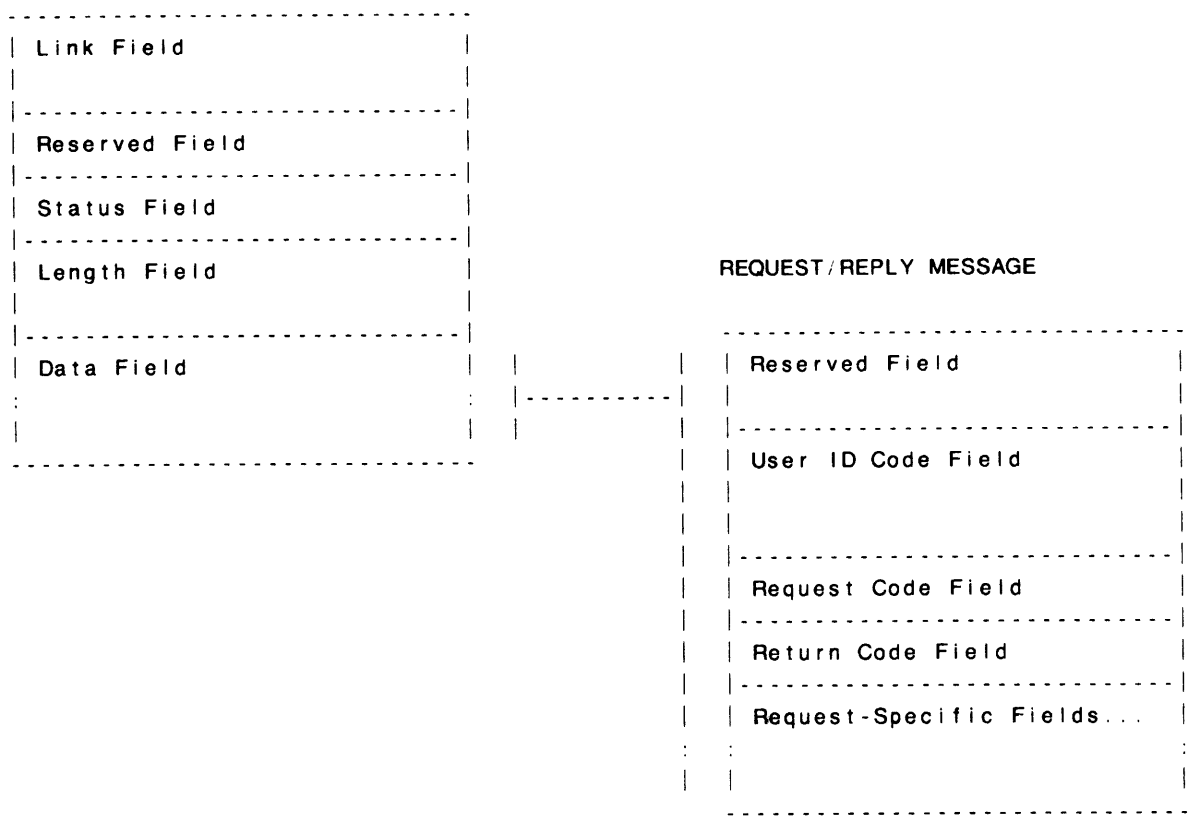


Figure 5-1: Encapsulation of Request/Reply Message in Message Buffer

Note that in link level controller mode, the EXOS 204 will buffer incoming packets (that pass address filtering) even if no receive requests have been submitted.

As shown by Figure 5-2, the EXOS 204 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 204.

The EXOS 204 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 a 32-bit user id field which is not interpreted by the EXOS 204 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.

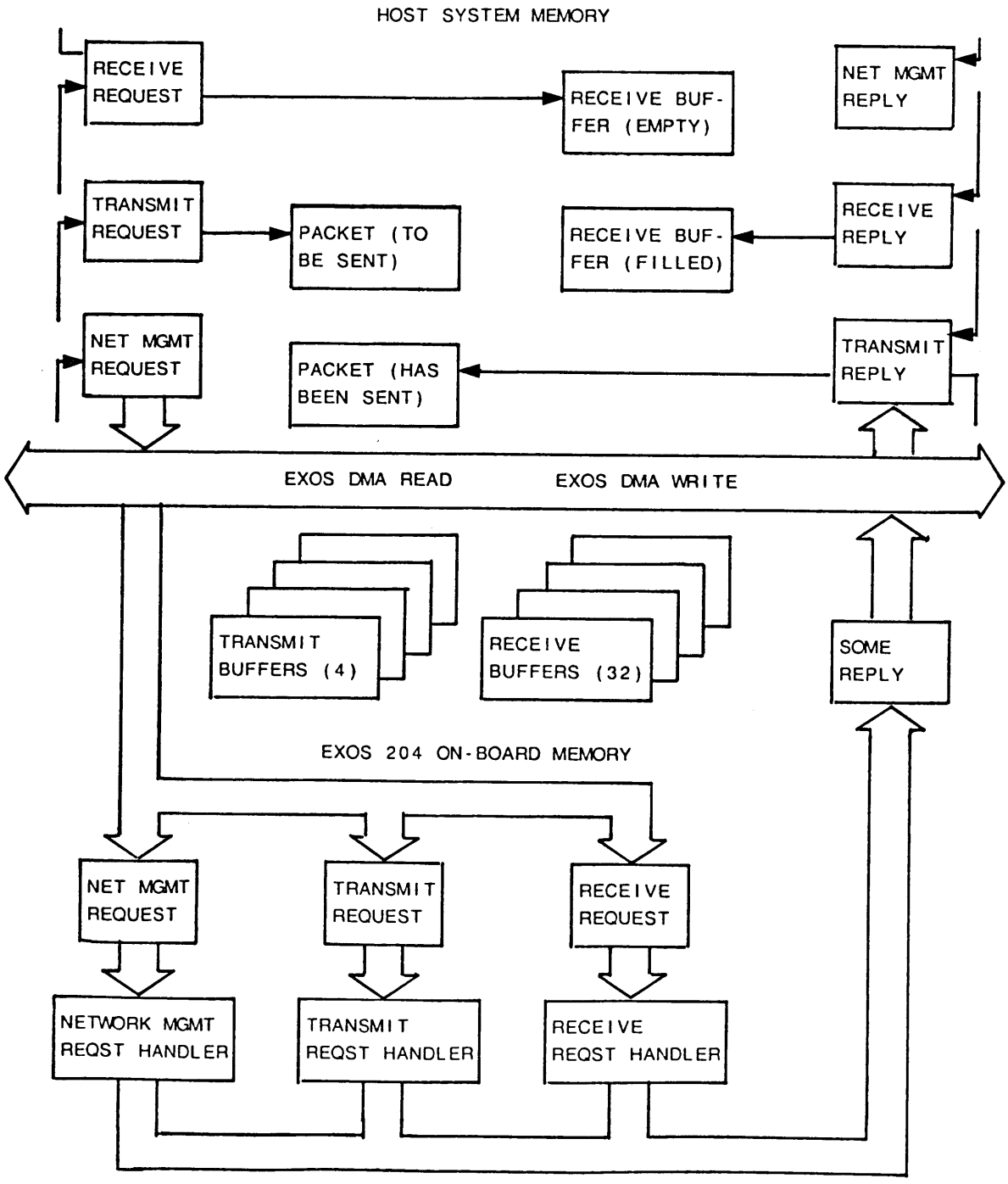


Figure 5-2: Link Level Controller Mode Request Processing Scheme

The remainder of this section specifies the format of the request/reply messages for each request. Where these requests map directly into NX 200 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 200 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 204 is not initialized in link level controller mode.

5.2. TRANSMIT Request/Reply Message

To transmit a packet on the Ethernet, host software sends a transmit request message to the EXOS 204. 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 204 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 204 enqueues transmit requests, and completes packet transmission without any intervention from the host. When the EXOS 204 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 204 is in off net mode (described in Section 7.3) then transmit requests will be enqueued, but will remain outstanding until the EXOS 204 is put back in an on net mode. If the EXOS 204 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 0EH (instead of 0CH). 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 204 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 204 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 200, 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	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

5.2.2. User Id Code Field

The user id code field is not interpreted by the EXOS 204, 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:

0CH	transmit.
0EH	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 in 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.
08H	(applicable for Version 2.0 transceivers only.) indicates the absence of SQE TEST signal during the Inerframe Spacing interval. This return code is OR-able with all other return codes except 40H and 0A1h. A jumper option is available to disable this check. (See Section 11.)
10H	transmission failed, excessive collisions.
20H	no Carrier Sense signal detected during transmission.
40H	transmission failed, transmit length not in range.
0A1H	failed, the EXOS 204 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 stations 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 204 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 204, 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

Host software receives a packet on the Ethernet by sending an Ethernet receive request message to the EXOS 204. This message contains pointers to a packet buffer in host memory. If the EXOS 204 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 204 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 204 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 204 receives and buffers the packet in its own memory. In link level controller mode, EXOS 204 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 204 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 EXOS-to-host message queue, the indicated buffer belongs to the EXOS 204 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 200, 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 204, and is returned unmodified in the reply message. It can be used to establish a correspondence between request and reply messages.

#	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	0DH	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)		

Figure 5-4: RECEIVE Request/Reply Message

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 in 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 204 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 1518 bytes long. Minimum size is 64 bytes, which will fit the shortest possible Ethernet packet.

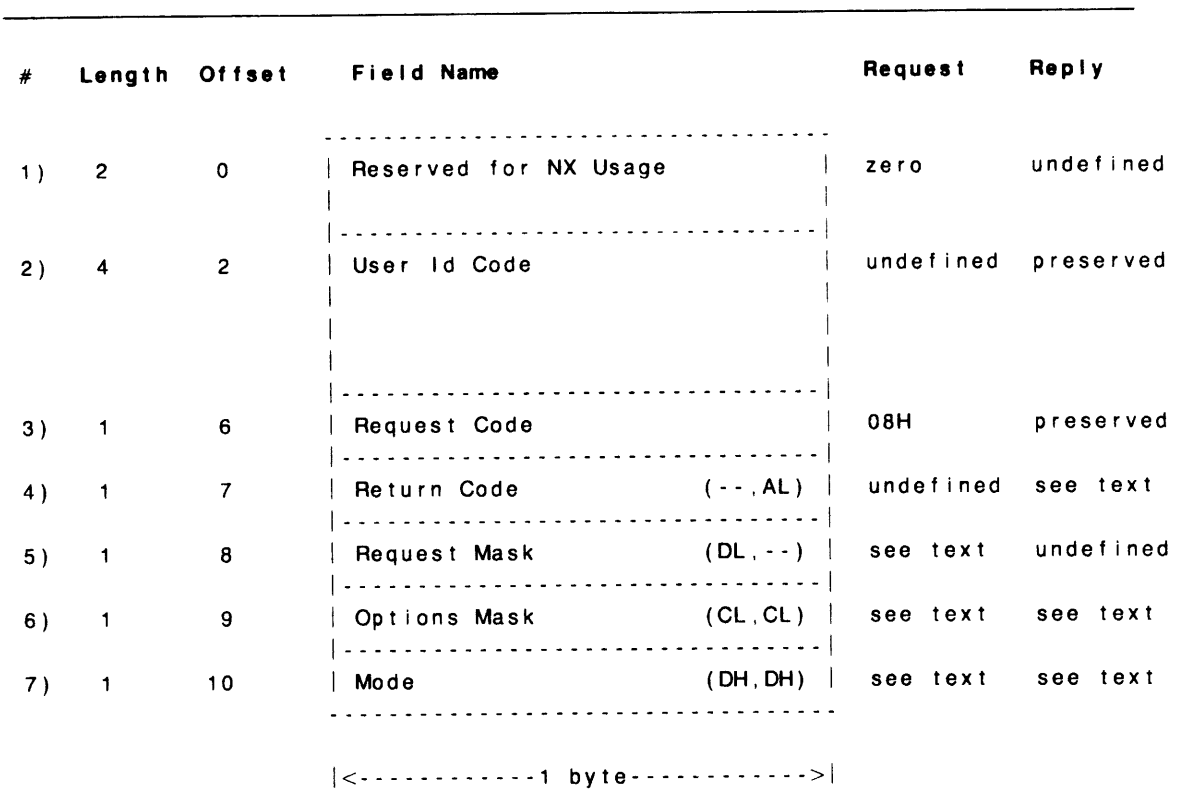
In the reply message, the buffer length field total returns the number of bytes actually received, plus 4 bytes for the CRC field. Note that the CRC value is not actually written back. Also, if the buffer supplied was smaller than the packet received, then the excess bytes are truncated, and the buffer 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 204 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

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.



#	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

Figure 5-5: NET_MODE Request/Reply Message

5.4.1. Reserved Field

The first field is reserved for use by NX 200, 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 204, 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 204 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 200, 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.
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

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.

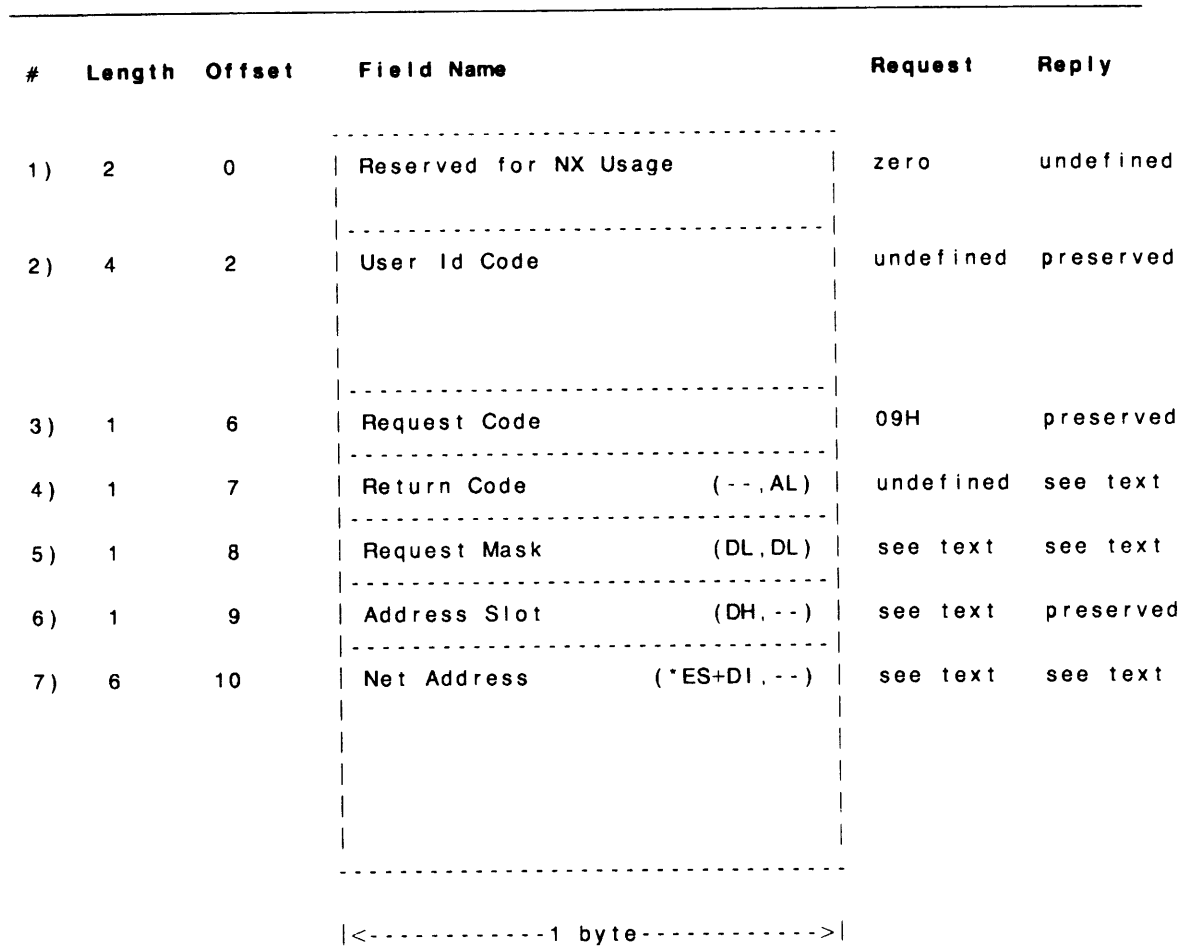


Figure 5-6: NET_ADDRS Request/Reply Message

5.5.1. Reserved Field

The first field is reserved for use by NX 200, 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 204, 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 204 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 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

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 200, 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		0AH	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

5.6.2. User Id Code Field

The user id code field is not interpreted by the EXOS 204, 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 0AH. 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 204 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

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 200, 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 204, 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.

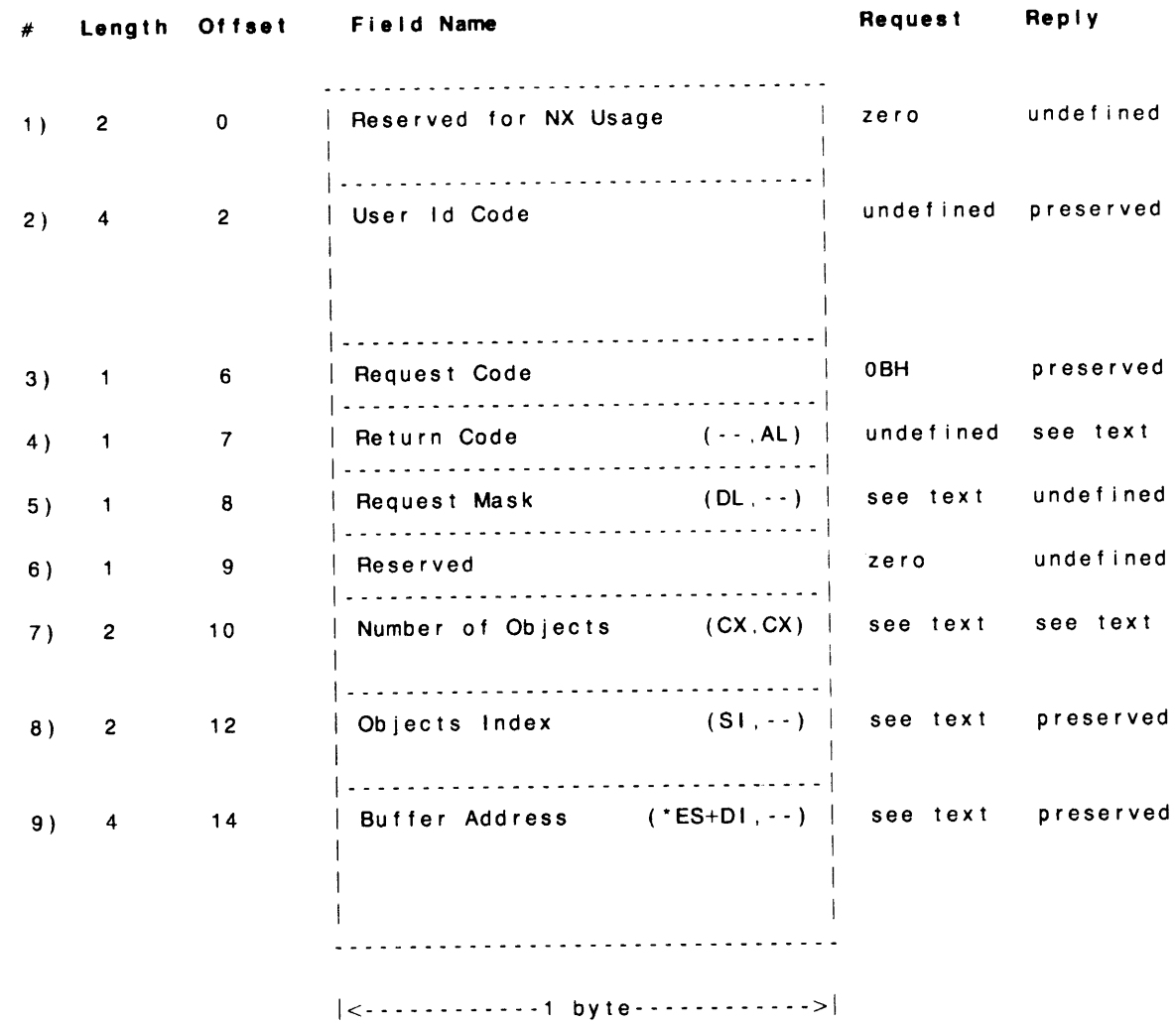


Figure 5-8: NET_STSTCS Request/Reply Message

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 204 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.

(blank page)

6. THE NX 200 PROGRAMMING ENVIRONMENT

This section provides information necessary to write higher-level software to run under the NX 200 kernel on an EXOS 204 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 200.

6.1. Overview

All programs for the Excelan Ethernet network processor board (EXOS 204) run on an Intel 80186 CPU under an EPROM-resident multi-tasking operating system kernel (NX 200). Programs can be written in any language for the 80186 and can be located anywhere in the memory available to the user. They can be down-loaded either from the host or over the network. The procedure for down-loading programs is described in Section 4.8 of this manual.

NX 200'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 200 has built-in drivers which provide suitable abstractions of the devices, so that programs developed for the EXOS 204 are independent of actual hardware implementation.

All functions of NX 200 are accessed by means of NX/200 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 80186 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 204, under the default configuration of NX 200. 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 200 can be changed at initialization time, with a corresponding effect on RAM allocation. The following paragraphs explain the use of EXOS 204 memory in detail.

6.2.1. Interrupt Vector Table

In the default configuration, NX 200 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.

6.2.2. Movable NX 200 Data Area

The movable NX 200 data area is the memory required for data structures which NX 200 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 200 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.

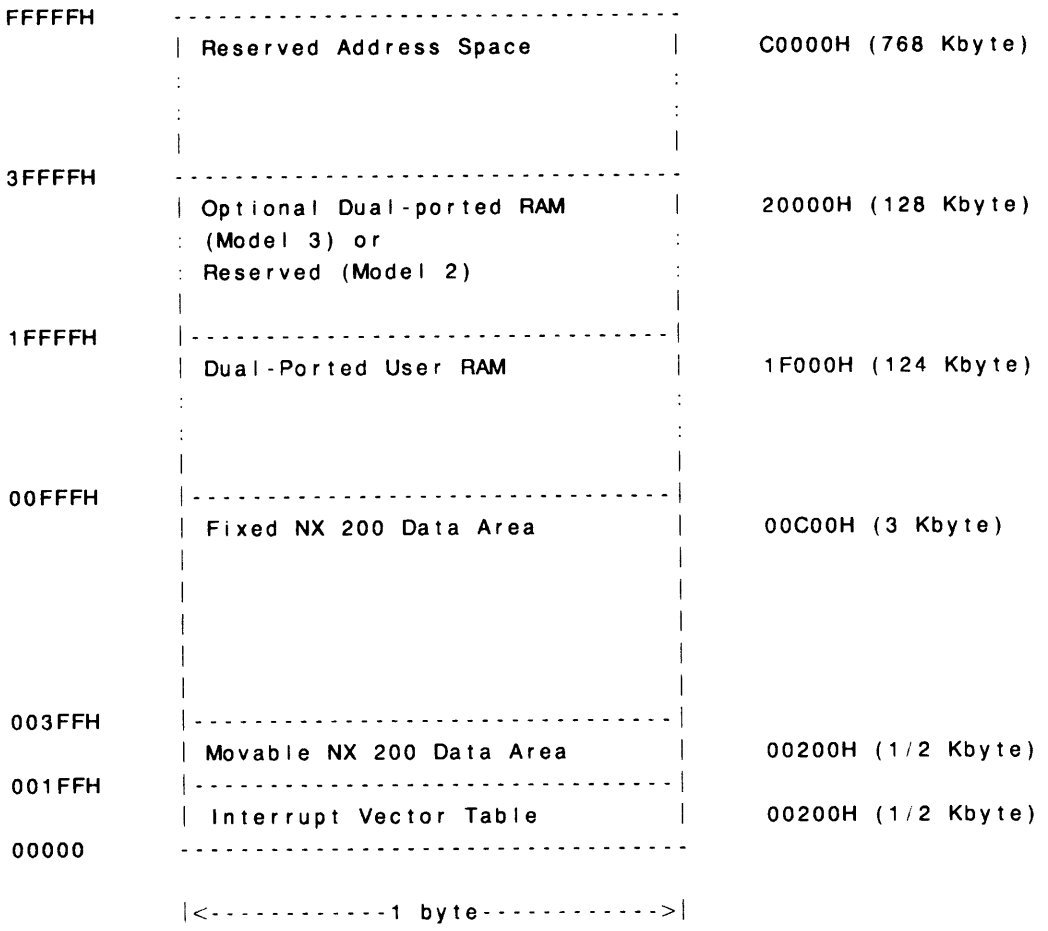


Figure 6-1: Default EXOS 204 Memory Allocation

If NX 200 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 200 Data Area

NX 200 uses this memory area for data structures that are not dependent on its configuration. It is always 3 Kbytes 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

128 Kbytes of RAM on the EXOS 204, from location 0 to 1FFFFH, is dual-ported between the 80186 CPU and the Ethernet controller hardware. Of this, 124 Kbytes between 1000H and 1FFFFH, 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 200 requires that all message buffers (used for communicating data between processes, host, and network) lie in dual-ported user RAM.

6.2.5. Optional Dual-Ported RAM

On the EXOS 204 Model 3 only, additional 128 Kbytes of RAM, from location 20000H to 3FFFFH is also available for the purposes of down-loaded software.

6.2.6. Reserved Address Space

The address range 20000H-FFFFFH on the Model 2, or 40000H-FFFFFH on the Model 3, is reserved. The effects of access to this area by user software are not defined.

Other than those described above, NX 200 imposes no restrictions on how memory is used. Users can link and load their programs in any manner they please. NX 200 does not define any buffer management services; users may choose the optimum scheme for individual applications.

6.3. Interrupt Types

The 80186 CPU provides 256 interrupt types, each of which corresponds to a 4-byte interrupt vector table entry.
NX 200 allocates these as follows:

0-31	reserved/dedicated by Intel.
32-63	device interrupts (used by NX 200).
64-95	NX 200 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 200 provides all interrupt service routines necessary for EXOS 204 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 200 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 200 (see Section 4.4.12).

6.4.1. Process Address Space

NX 200 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 80186. Thus each process can independently choose its own direct address space. The 80186 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 200 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 200 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 200 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 200 creates the first process, and allocates its stack within the fixed NX 200 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 200 does not maintain any separate system stack for a process. When NX 200 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 200 interrupt service routines.

6.4.4. Process Scheduling

Four parameters visible to user software drive NX 200'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 200 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 200 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 200 call or by an implicit side effect of some other NX 200 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 200.

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 200 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 200 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

receive 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 200 (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 200 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 200 uses the link field to chain messages. They must reside within the address range 0-0FFFFH in EXOS 204 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 200 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 200 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 200. In general the user programs should not disable interrupts.

6.7. System Mailboxes

Certain NX 200 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 200 during initialization and are not included in the number of user mailboxes specified by the configuration of NX 200. The set of mailbox-ids for regular mailboxes is disjoint with the set of mailbox-ids for system mailboxes. NX 200 designates the following system mailbox-ids:

0001H Host
0009H Ethernet

Request messages are sent through the system mailbox using the MLBX_SEND call. After completing the request, the system service returns the reply message to the reply mailbox specified in the request message. After sending a request message, a process can continue to run or wait until the reply message is returned. The request message, once sent, should not be modified until the reply message is received. The user process can receive the reply message from the reply mailbox using the MLBX_RECV call.

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.

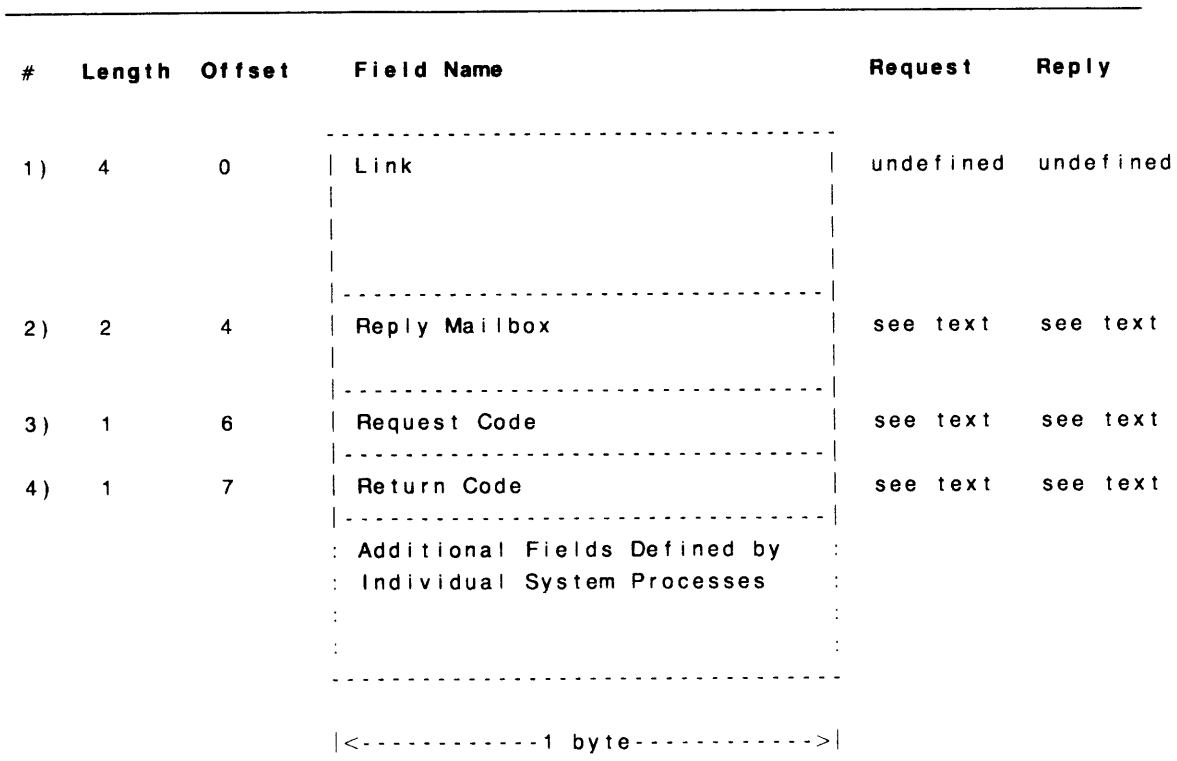


Figure 6-2: Standard Header for System Messages

6.7.1. Link Field

The link field is required by NX 200 at the beginning of all messages. Its request and reply values are both undefined. NX 200 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 200'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 200'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.

(blank page)

7. THE NX 200 ETHERNET INTERFACE

The NX 200 Ethernet interface consists of two parts: a system process which sends and receives packets, and several NX 200 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 200 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 200 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 204 hardware.

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 200 at the beginning of all messages. Its request and reply values are both undefined. NX 200 uses this field for chaining messages.

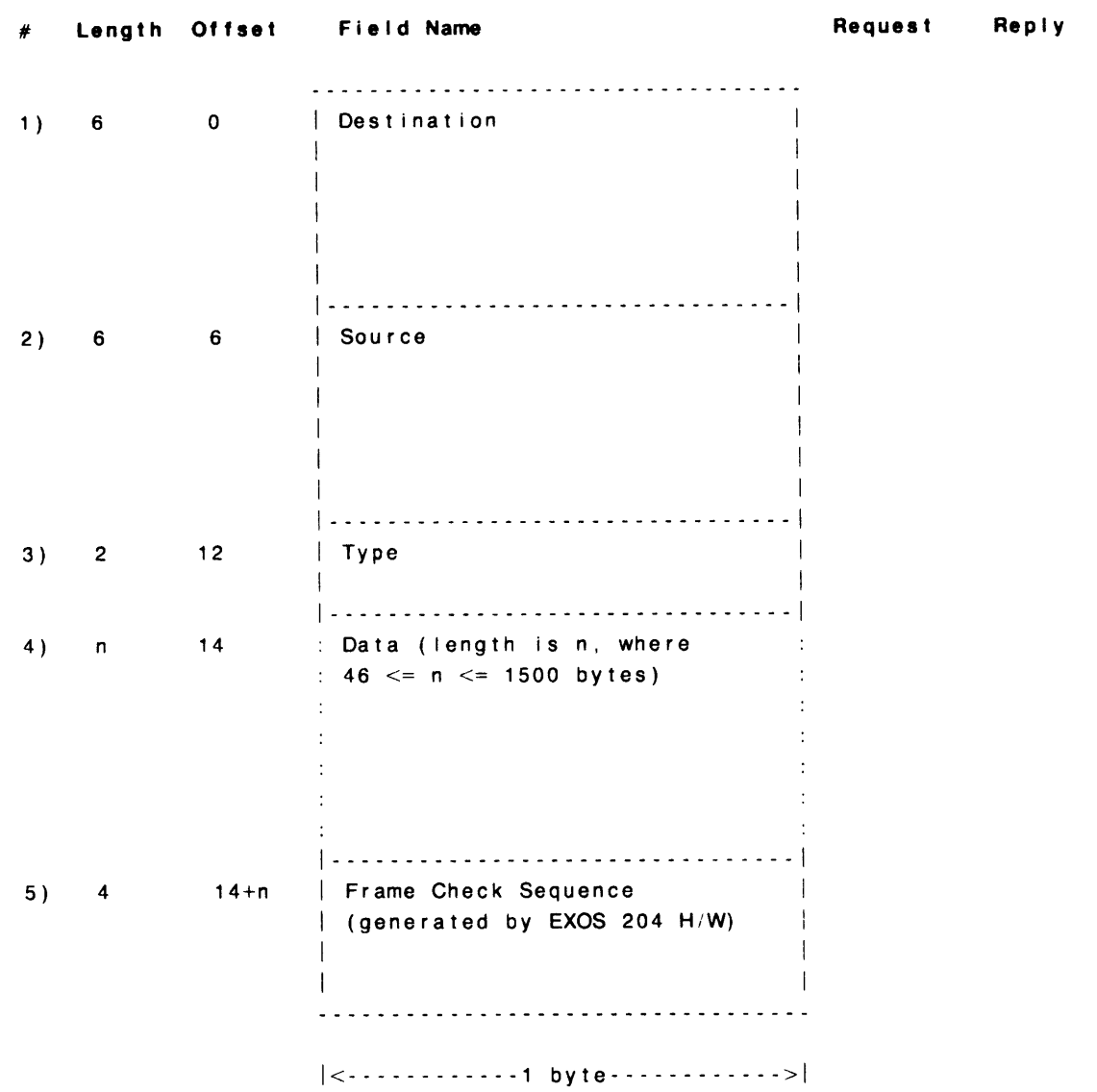


Figure 7-1: Ethernet Packet Format

7.1.2. Reply Mailbox Field

The reply mailbox field identifies the mailbox to which the reply 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.

7.1.3. 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.

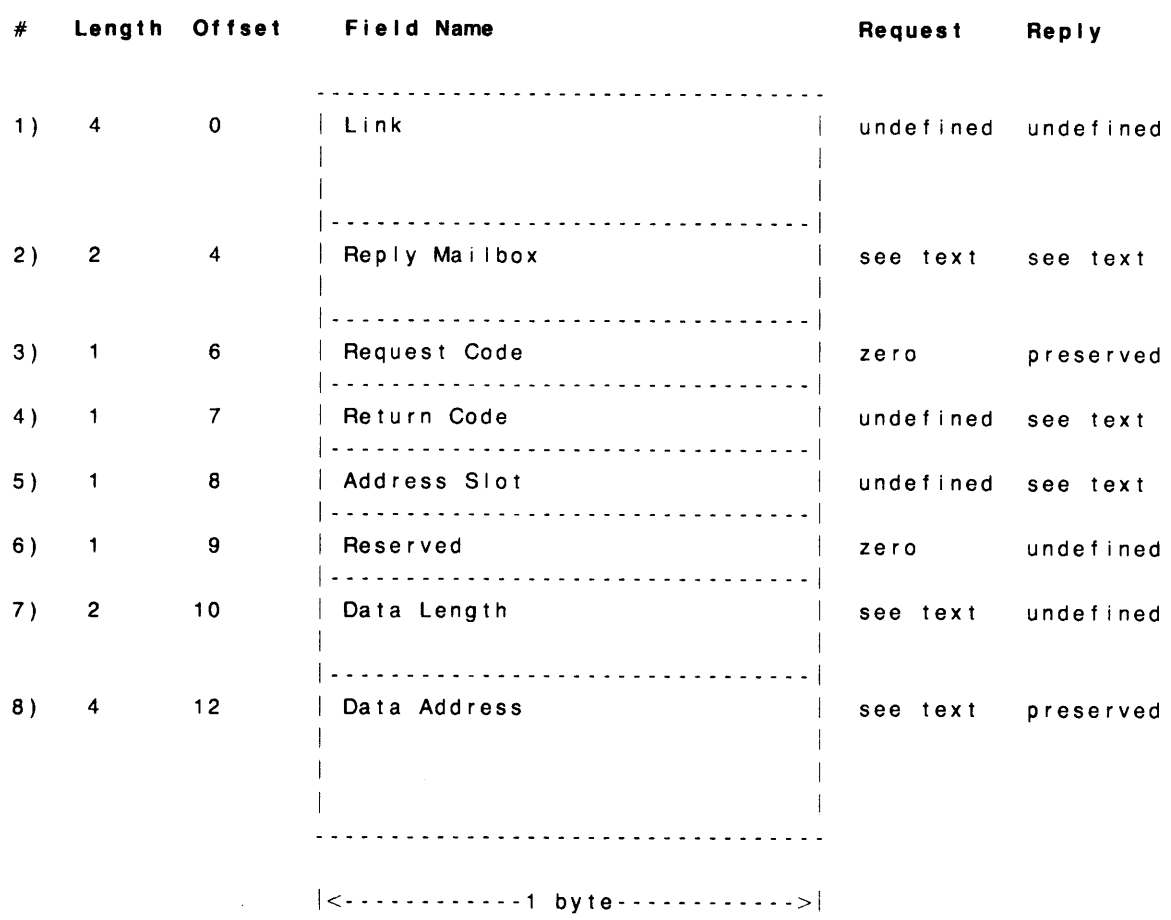


Figure 7-2: Ethernet Transmit Request/Reply Message

7.1.4. Return Code Field

The return code field value in 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.

08H (applicable for Version 2.0 transceivers only.) indicates the absence of SQE TEST signal during the Inerframe Spacing interval. This error code is OR-able with all other return codes except 40H. A jumper option is available to disable this check.

10H transmission failed, excessive collisions.

20H no Carrier Sense signal detected during transmission.

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 204 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 204 requires that the packet start at an even boundary and 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 204 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 204 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 200. This prevents any spurious network bootstrap messages from interfering with successfully-installed protocol software on the EXOS 204.

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.

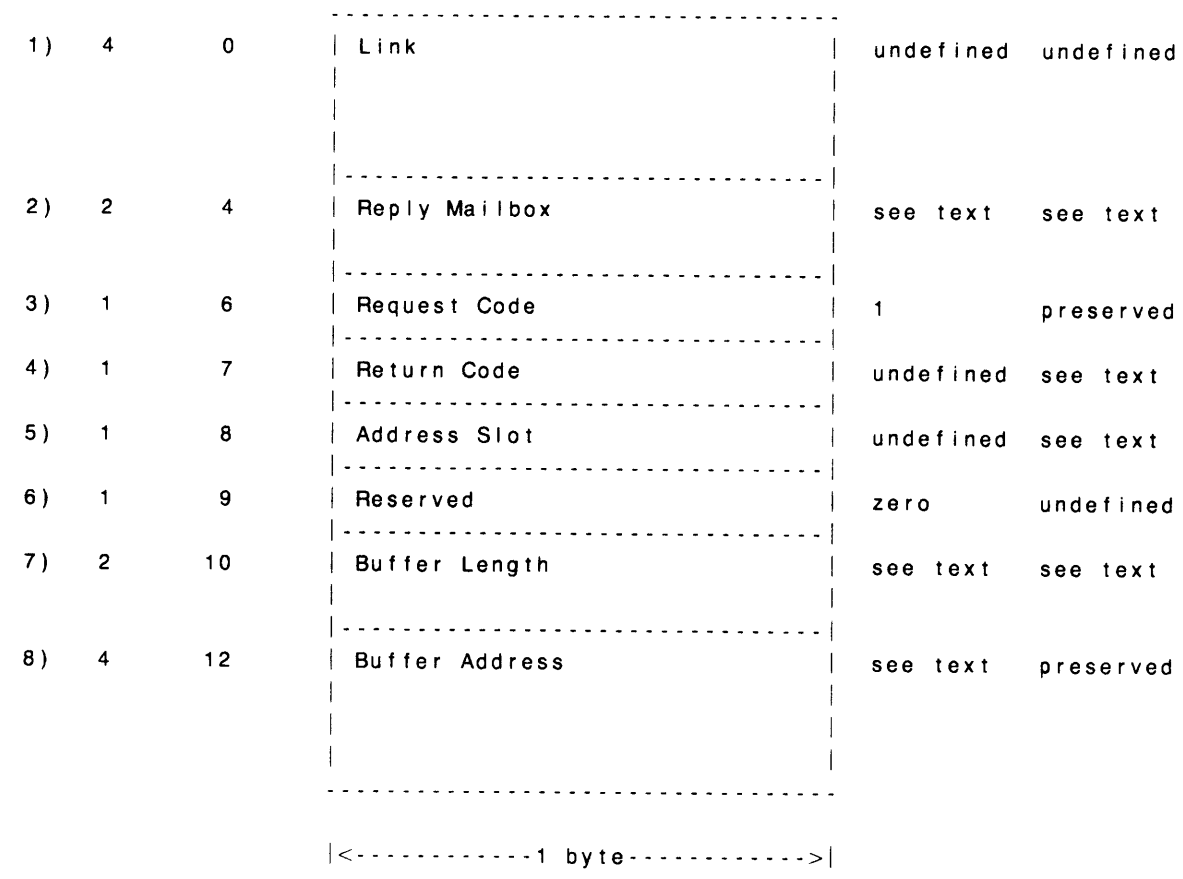


Figure 7-3: Ethernet Receive Request/Reply Message

7.2.1. Link Field

The link field is required by NX 200 at the beginning of all messages. Its request and reply values are both undefined. NX 200 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 in 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.

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 1518 bytes long. Minimum size is 64 bytes, which will fit the shortest possible Ethernet packet.

In the reply message, the buffer length field returns the number of bytes actually received, plus 4 bytes for the CRC field. Note that the CRC value is not actually written back. Also, 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 204 requires that the buffer start at an even boundary and lie entirely within the address range 1000H-1FFFFH. 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 204 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 204 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 204 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 204 is connected to the Ethernet. All packets are received regardless of their destination address.

When the EXOS 204 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 200, 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 200, are given in Section 4.4.14.

Every EXOS 204 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 204 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_ADDRS 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 204 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 204 supports network management functions by gathering statistics on network operations. The statistics appear to the user as an array of longword (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 204 resets all statistics objects - otherwise they are continuously maintained.

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 SQE TEST error - a 32-bit counter that counts the number of those transmitted frames which encountered heartbeat absence error. (This object applicable for Ethernet Version 2.0 transceivers only. A jumper option is available to disable this feature. (See Section 11.)

3 Undefined - reserved for future use.

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.

(blank page)

8. THE NX 200 HOST INTERFACE

User software on the EXOS 204 communicates with the host through a system process in the NX 200 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 200 also provides the MEM_READ and MEM_WRITE calls which access shared Unibus memory directly. This section describes these facilities as seen by a process on the EXOS 204. 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 204. 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 204 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 200 at the beginning of all messages. Its request and reply values are both undefined. NX 200 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.

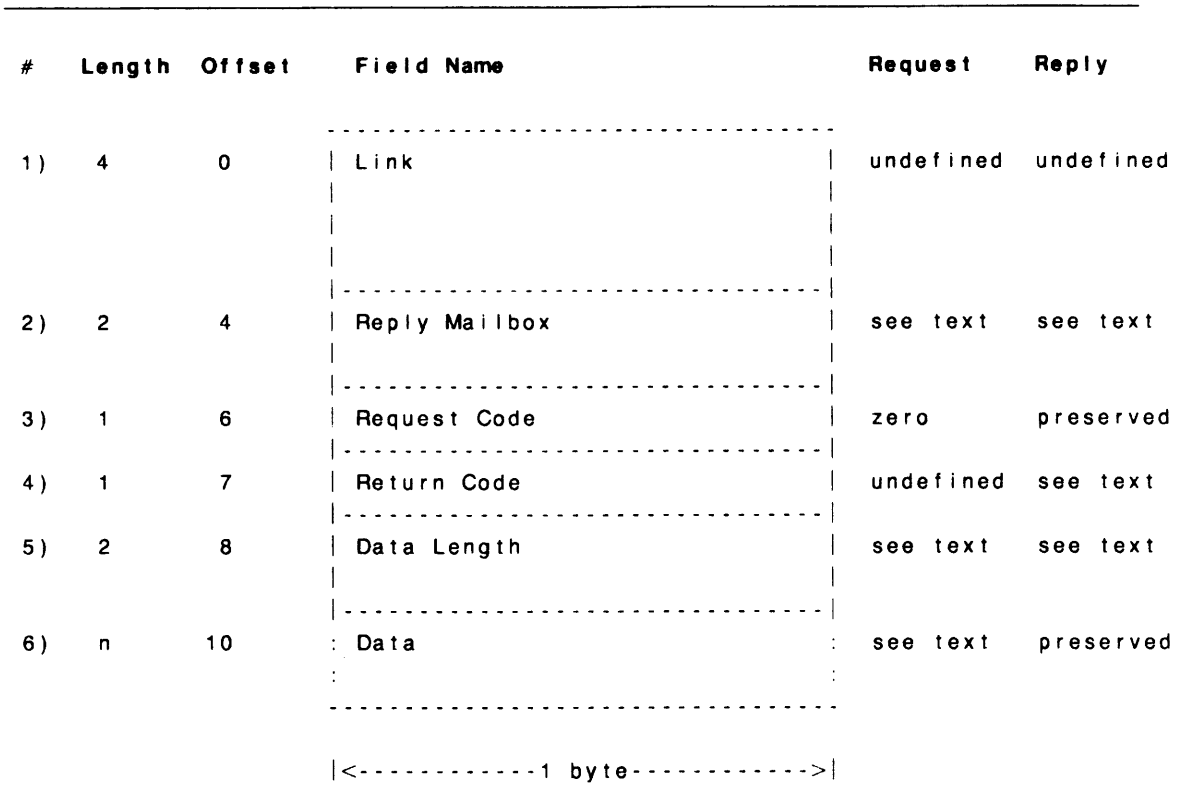


Figure 8-1: Host Transmit Request/Reply Message

8.1.4. Return Code Field

The return code field value in 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 200 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.

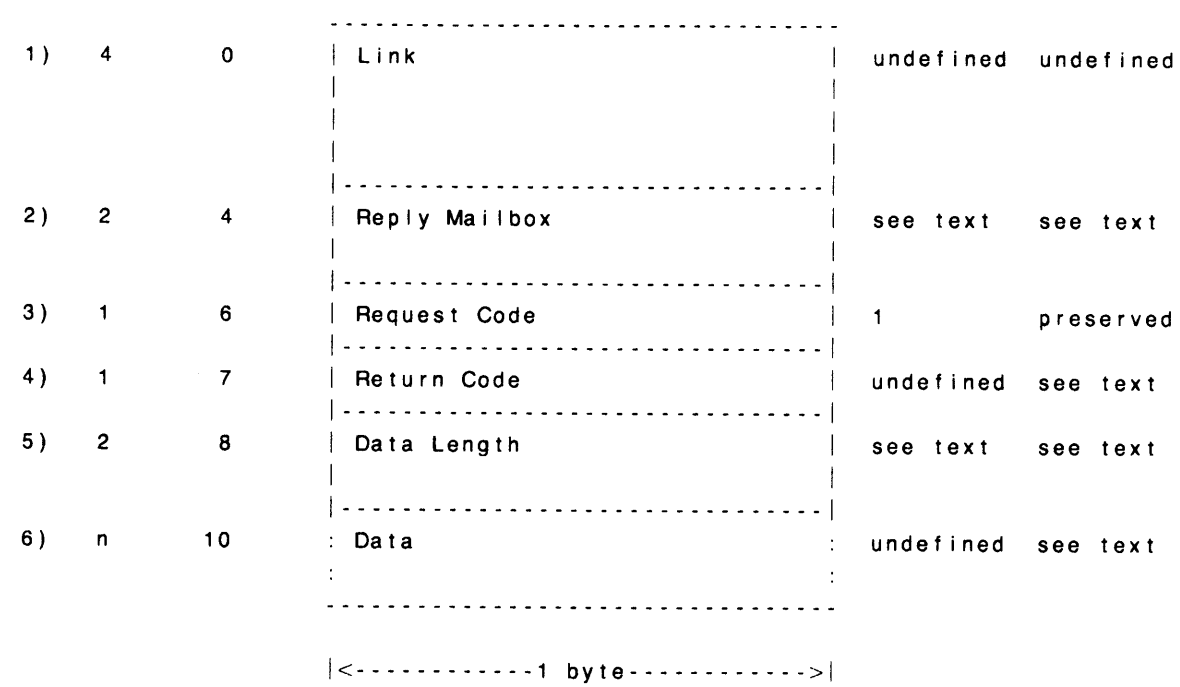


Figure 8-2: Host Receive Request/Reply Message

8.2.1. Link Field

The link field is required by NX 200 at the beginning of all messages. Its request and reply values are both undefined. NX 200 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 in 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 204 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 200 will apply any conversion needed for the byte string data type.

8.3. Direct Access to Host System Memory

The EXOS 204 accesses Unibus memory by mapping part of its own CPU's address space into Unibus memory addresses. This is the underlying mechanism which NX 200 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 200's MEM_READ and MEM_WRITE calls.

These calls take an address in EXOS 204 memory, an address in host memory, and a data transfer length. NX 200 performs the appropriate mapping and executes the transfer. If the host data order conversion option is enabled, NX 200 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 204, NX 200 provides calls which convert data between the host system's ordering and the 80186 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 204.

When the host data conversion option is enabled, NX 200 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 204 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 200 KERNEL CALL REFERENCE

This section defines the specific format and usage of NX 200 kernel calls. User software running on the EXOS 204 should access all NX 200 services through these requests. For more information about the function of NX 200 kernel calls, see Sections 6, 7, and 8.

Processes request NX 200 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 200 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_ADDRS	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 204 version number.

The remainder of this section describes the NX 200 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 (0FFFFH) 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 (0FFH) 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.

0F0H 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 memoryi; this area should not be used for any other purpose. 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 (0FFFFH) 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 (0FFH) 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 (0FFFFH), 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 then 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. 0FFFFH 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 0-0FFFFH. The first field of the message should be a 32-bit link field available for use by NX 200. 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 (0FFFFH) 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 (0FFFFH) indicates a null message.

ES: segment part of the message address if the call is successful, otherwise undefined. NX 200 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 204 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 200, 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_ADDRS **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. This number must be less than or equal to 64K-16.

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 204 memory.

ES: segment part of the address in EXOS 204 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 204 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.

Note that this call may potentially block the user process, and as a result, all locks in effect would be removed.

MEM_WRITE **INT 83**

Parameters:

CX: number of bytes to be written. This number must be less than or equal to 64K-16.

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 204 memory.

ES: segment part of the address in EXOS 204 memory.

Results:

AL: completion code:

0 successful.

AH: undefined.

Description:

This call copies the specified number of bytes from the specified address in EXOS 204 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.

Note that this call may potentially block the user process, and as a result, all locks in effect would be removed.

VERSION **INT 84**

Parameters:

none.

Results:

AL: always 0.

AH: undefined for NX Version numbers less than 4.0; EXOS context otherwise.

CX: version of NX 200.

DX: version of the EXOS 204 hardware.

Description:

This call returns the version number of the EXOS 204 hardware and NX 200. 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.

If the NX 200 version number returned in CX is less than 4.0, then AH contents are undefined. If the NX 200 version number returned in CX is equal to or greater than 4.0, then AH contains the EXOS context. For EXOS 204 the context value is 04.

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 80186 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 80186 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.

(blank page)

10. INITIALIZING AND DOWN-LOADING FROM THE ETHERNET

The EXOS 204 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 200 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 204:

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 204 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 204) 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 204 states, shown to the left of the diagram, are as follows:

RESET denotes that the EXOS 204 has been reset, but has not yet attempted a network bootstrap.

FIND REPLY WAIT denotes that the EXOS 204 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 204 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 204 has received a SELECT reply message, and is now awaiting a command request message from the selected boot server.

EXECUTE denotes that the EXOS 204 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 204 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 204 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 204 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 204 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.

2) The EXOS 204 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 204, down-load code, up-load its memory contents, and begin execution. For each request, the EXOS 204 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 204's point of view:

1) The EXOS 204 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

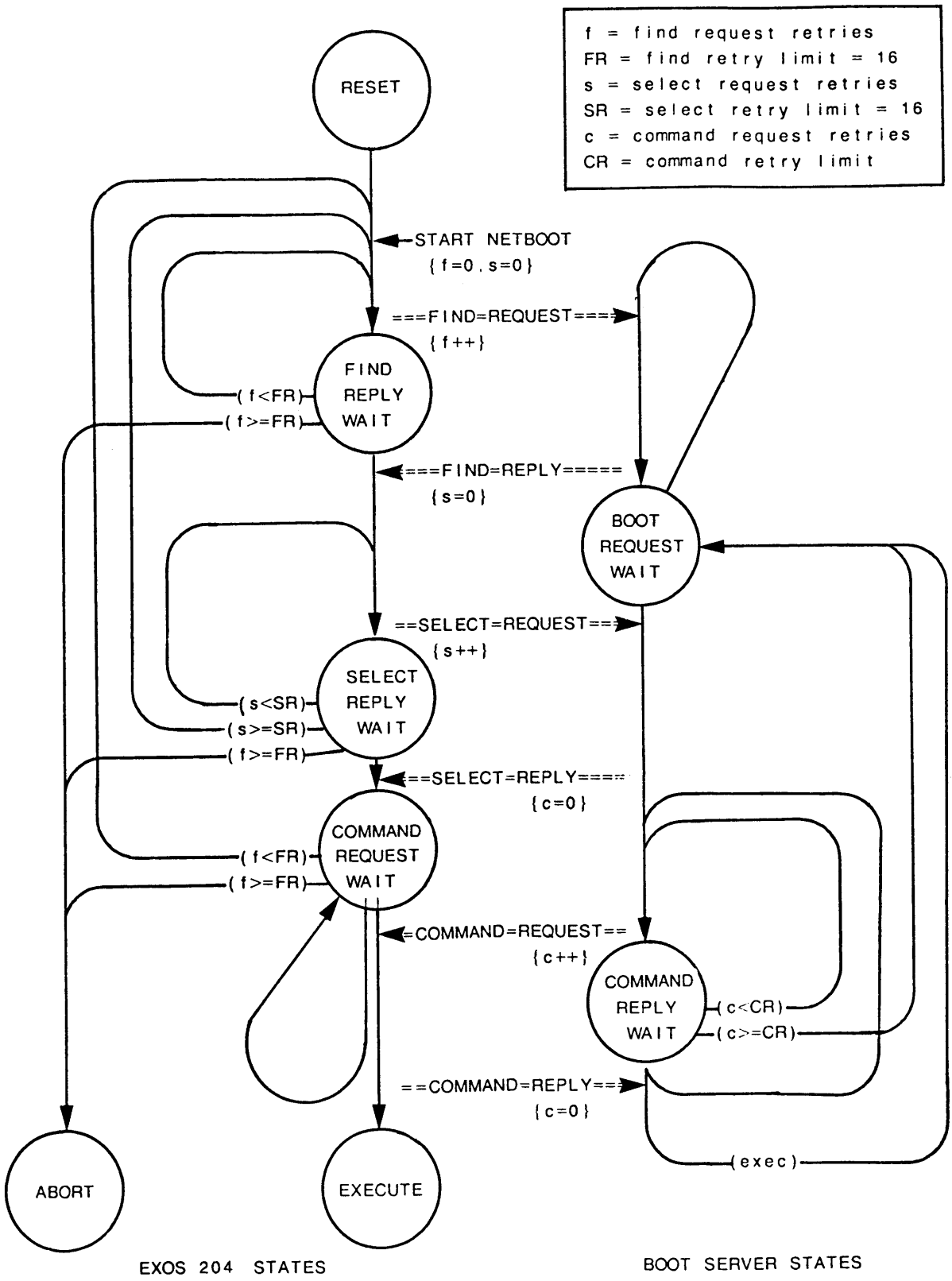


Figure 10-1: State Diagram of Network Bootstrap Protocol

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 204 supports.

b) the number of buffers available on the EXOS 204 to receive incoming network bootstrap COMMAND request messages.

c) the length of the buffers described above.

d) the Ethernet address of the EXOS 204 (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 204 will wait for a reply message before re-trying or giving up.

g) a configuration message, which describes the current configuration of the EXOS 204. 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 204 enters the FIND REPLY WAIT state.

2) Before its FIND reply timeout expires, the EXOS 204 should receive a FIND reply message from at least one qualified bootstrap server. If more than one is received, the EXOS 204 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 204 should wait for the boot server's SELECT reply message.

Immediately upon receiving a legitimate FIND reply message, the EXOS 204 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 204 client. The SELECT request message contains exactly the same information as the FIND request message, except possibly for the timeout parameter. The EXOS 204 specifies its current effective timeout value in this field. After sending the SELECT request message, the EXOS 204 enters the SELECT REPLY WAIT state.

3) Before its SELECT reply timeout expires, the EXOS 204 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 204 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 204 enters the COMMAND REQUEST WAIT state.

4) Before its COMMAND request timeout expires, the EXOS 204 should receive a COMMAND request message from the selected bootstrap server. When a

command arrives, the EXOS 204 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 204 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 204 is concerned) and the EXOS/204 proceeds to execute the code which has presumably been down-loaded.

While the description above specifies exactly how the EXOS 204 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 204 provides just one receive buffer for processing net boot commands. Therefore it assumes that only one command may be outstanding. Future releases of NX 200 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:

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 204 will ignore messages from other sources. Another general principle the EXOS 204 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 204 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 204 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 204 will timeout. Depending on circumstance, it may then abort or retry, possibly entering a different state. The EXOS 204 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 204 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 204 enters the ABORT state. The EXOS/204 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 204 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 204 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 204 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 204'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 204 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 204 generates a message ID field for both FIND request and SELECT request messages. This ID guards the EXOS 204 against spurious message reception up to the point that a network bootstrap session is established. The EXOS 204'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 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 204 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 204 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 204 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 200. Network bootstrap servers must obey these conventions when transmitting messages to the EXOS 204, and should observe them when interpreting messages received from the EXOS 204.

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 80186 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 204. When implementing a bootstrap server based on a different CPU architecture, programmers should be careful to observe this ordering.

Note that the EXOS 204 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 204 and its host processor. The rest of the CONFIGURE request message, and all other messages, will still be interpreted according to 80186 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.

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 204 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.

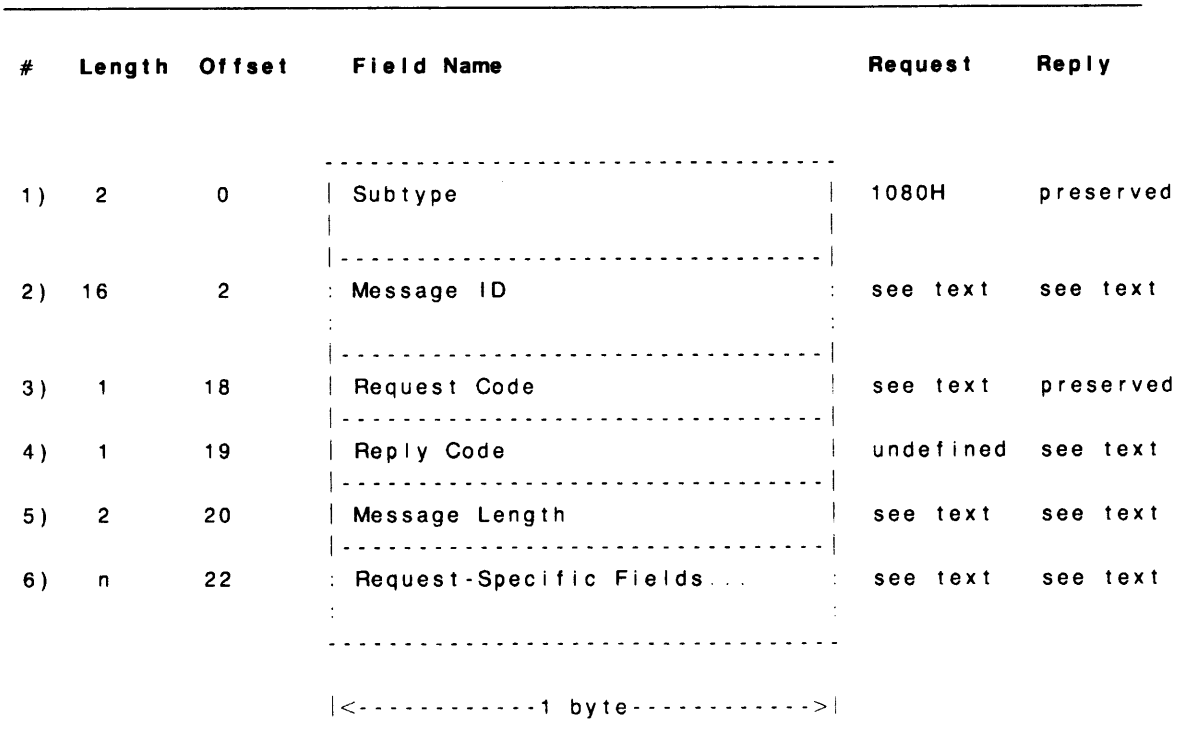


Figure 10-2: Network Bootstrap Protocol Request/Reply Message Header

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 204 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 204 in the destination address field.

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

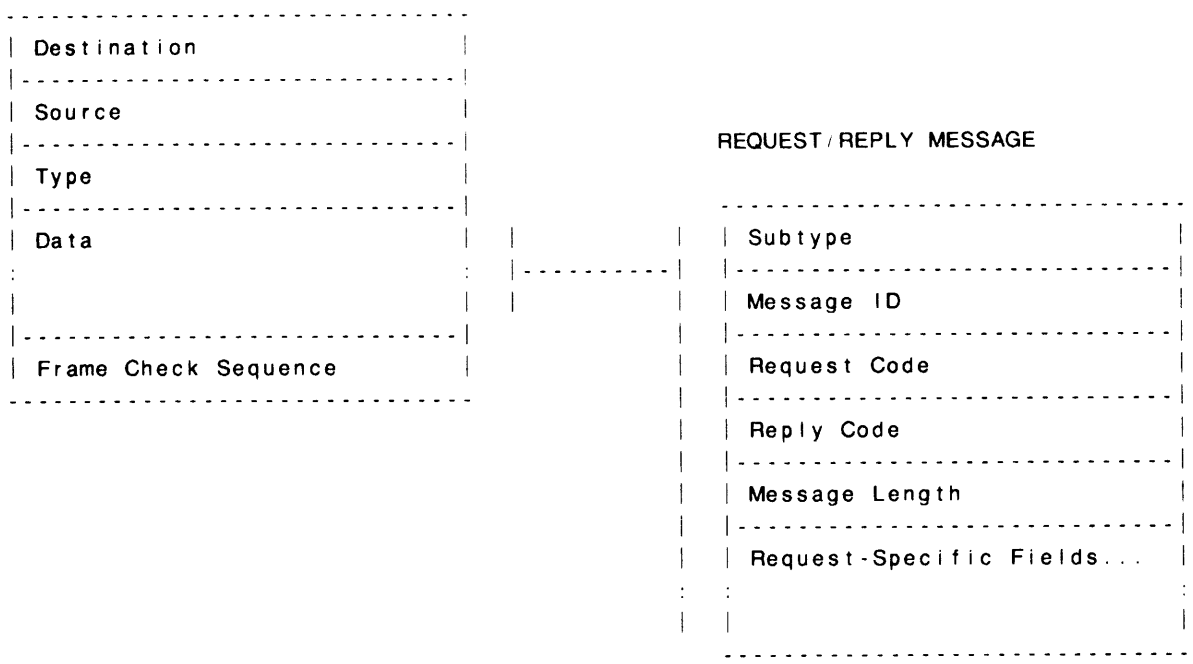


Figure 10-3: Encapsulation of Request/Reply Message

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 204 itself, this is equivalent to storing the 16-bit value 1080H in the 80186 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 204 broadcasts the FIND request message to identify bootstrap servers, which return a FIND reply message to the client's physical address. The EXOS 204 then sends the SELECT request message to the physical address of a boot server, telling it to bootstrap the EXOS 204. The boot server acknowledges this with a SELECT reply message. The following paragraphs describe the individual fields in detail. 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 204 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 204 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 204. 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 boot protocol version is 1 for all EXOS boards. 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 204 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 204's receive buffer. This determines the maximum size COMMAND request packets the EXOS 204 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 204 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 204 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 204 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 204 observes when waiting for some message from the boot server. It is specified in milliseconds, but the EXOS 204 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 204 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 204 performs a FIND request retry. Therefore the EXOS 204 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 204 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 0FFH selects the current value. If the value specified is

0, then the EXOS 204 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 204, 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.

Also included in this field is the EXOS Context information which indicates the EXOS 200 series controller board in use at the requesting end. For the EXOS 204 the Context value is 04. A boot server must check this field to assure that the value indicates a product that the boot server is designed to serve.

10.6. DOWNLOAD Request/Reply Message

The bootstrap server can use the DOWNLOAD request message to down-load code and data to the EXOS 204'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 204 will preserve this entire field's value.

The **request code field's** value for the DOWNLOAD request is 0. The EXOS 204 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 200, no copy done.
- A1H 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.

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 204 memory.

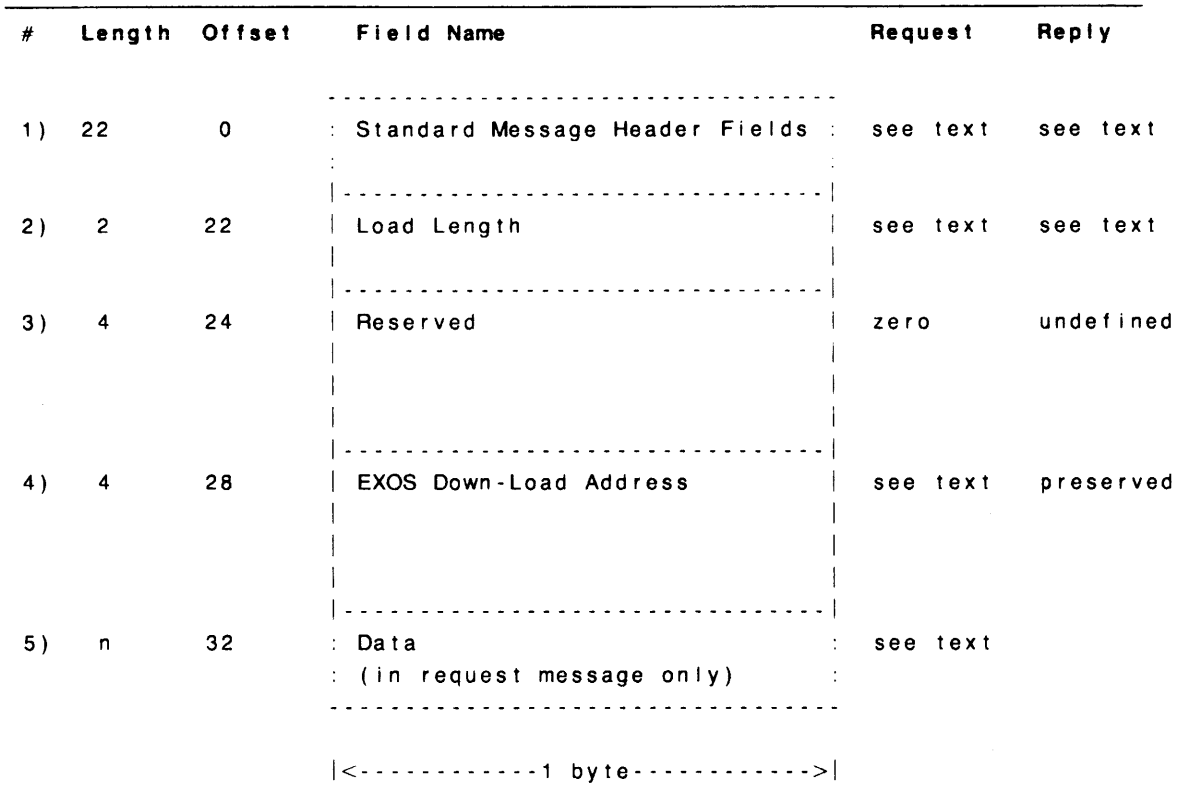


Figure 10-5: Network Bootstrap DOWNLOAD Request/Reply Message

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

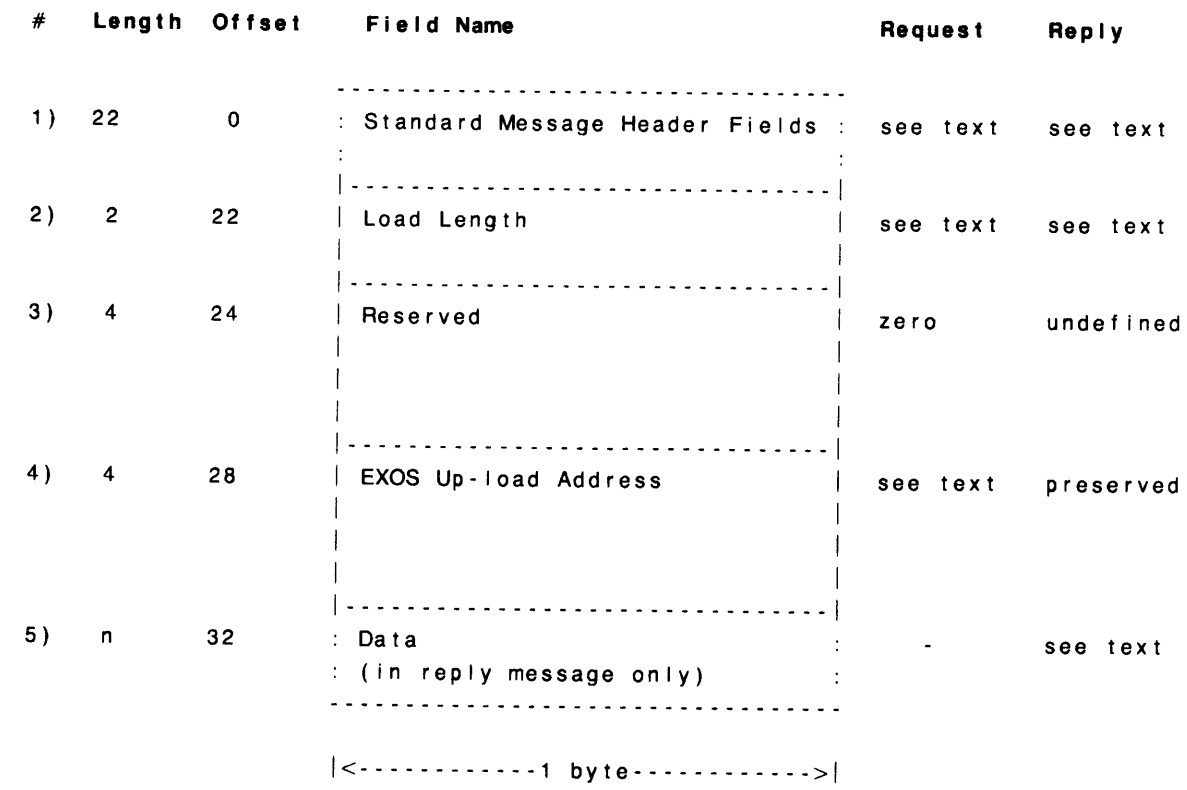
The EXOS down-load address field specifies the address in EXOS 204 memory to which the data should be transferred. Note that, as with all addresses referring to locations in EXOS memory, this should be a segmented address in the 8086 style. Its value is preserved in the reply message.

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 204'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

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 204 will preserve this entire field's value.

The **request code field's** value for the UPLOAD request is 1. The EXOS 204 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 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 204's memory. In the reply message, this field returns the number of bytes actually read from EXOS 204 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 204 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 204'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 204 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.

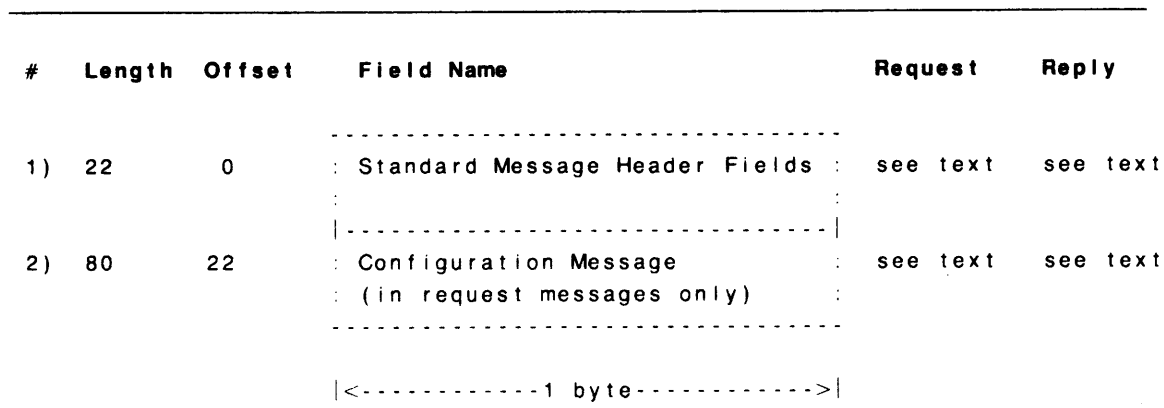


Figure 10-7: Network Bootstrap CONFIGURE Request/Reply Message

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 204 will preserve this entire field's value.

The **request code field's** value for the CONFIGURE request is 3. The EXOS 204 returns the same value in the 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 204. Once the EXOS 204 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.

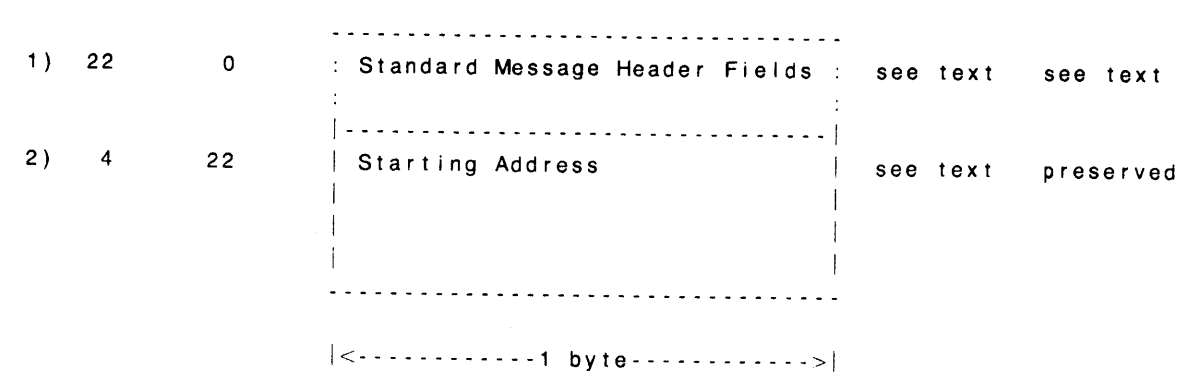


Figure 10-8: Network Bootstrap EXECUTE Request/Reply Message

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 204 will preserve this entire field's value.

The **request code field's** value for the EXECUTE request is 2. The EXOS 204 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 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 204 implementation are hidden by NX 200, ensuring that high-level software written for the EXOS 204 will be portable to future products. This section provides all necessary hardware interface and configuration information. Theory of operation is deliberately omitted.

> **NOTE**
>
> Before installing the EXOS 204, the jumper connecting the pins CA1 and CB1 on the Unibus backpanel must be removed. Otherwise, the EXOS 204 will not function properly.

11.1. Access to EXOS 204 Components

Appendix A shows the EXOS 204's layout, and the locations of accessible components. For development purposes, the following components are socketed:

80186 CPU
two 16K EPROMs

The EXOS 204 provides several jumpers to select addresses and options. Tables 11-1 and 11-2 contain quick reference to jumper functions, by number. Subsequent sections explain the jumpers in more detail.

The EXOS 204 includes four Light Emitting Diodes (LEDs) to communicate status information. These LEDs are designated as DS1, DS2, DS3, and DS4. DS1-DS3 located in adjacent positions at the top center of the board, seen from the component side, and can easily be seen while the board is installed. DS4 is located towards the upper left hand corner of the board. Figure 11-1 briefly shows their individual locations and functions. Subsequent sections explain the LEDs in more detail.

11.2. Unibus Interface

The EXOS 204 Ethernet Front-End Processor is built on a single quad-sized (10.44" by 8.9") Unibus board which occupies one Unibus SPC slot. It presents one DC load on the Unibus.

11.2.1. Unibus Compliance

The EXOS 204 conforms to Unibus specifications as a 16-bit bus master. Compliance is as follows:

8-bit or 16-bit transfers,
18-bit addressing,
bus-vectored interrupts.

11.2.2. Unibus Memory Access

The EXOS 204 generates 18-bit memory addresses to access the entire 256 Kbytes of Unibus memory.

Note that the EXOS 204's own memory is not accessible from the Unibus.

Table 11-1: Quick Reference to Jumper Options

jumper	function (when jumper is installed)			factory setting
J2	Invoke the EPROM resident Monitor program on power-up or reset			Absent
J3	DMA with burst capability presend			Installed
J4	Disable SQE (Heartbeat) check			Installed
J6	Used in conjunction with J9; specifies available RAM:			Absent
	J6	**J9**	**RAM**	
	Absent	Absent	128K (factory setting)	
	Installed	Absent	256K	
	Absent	Installed	Reserved	
	Installed	Installed	Reserved	
J7	Boot from network			Absent
J9	See description under J6.			Absent
J12	27256 user EPROMS			Absent
J13	2764 or 27128 user EPROMs			Installed
J14	27256 Kernel EPROMs			Absent
J15	2764 or 27128 Kernel EPROMs			Installed
J16	(3 jumpers) bits 10-12 of I/O port address (Installed = 0; absent = 1.)			Absent J16-2; others installed.
J17-1	Select DMA burst length			Installed
J17-2	Select DMA burst length			Absent
	J17-1	**J17-2**	**burst-length**	
	Installed	Installed	2	
	Installed	Absent	4	
	Absent	Installed	8	
	Absent	Absent	16	
J17-3	Enable bus timeout			Absent
J18	(8 jumpers) bits 2-9 of I/O port address			All installed (0)
J19	Configure Unibus interrupt levels**			
J20	Configure Unibus interrupt levels**			

** See Table 11-3 for details.

11.2.3. Unibus I/O Access

The EXOS 204 can access the full 8K Unibus I/O address space. However, it does not normally generate any I/O commands, unless requested by user software.

The EXOS 204 presents two read/write I/O ports to the Unibus. Their functions are documented in Section 4.1. Port A's address is fully jumper-selectable, at any even word address in the I/O address range 760000 to 777774 (Octal) corresponding to 3FE800 to 3FE802 (Hexadecimal). Port B's address is the address of port A plus 2.

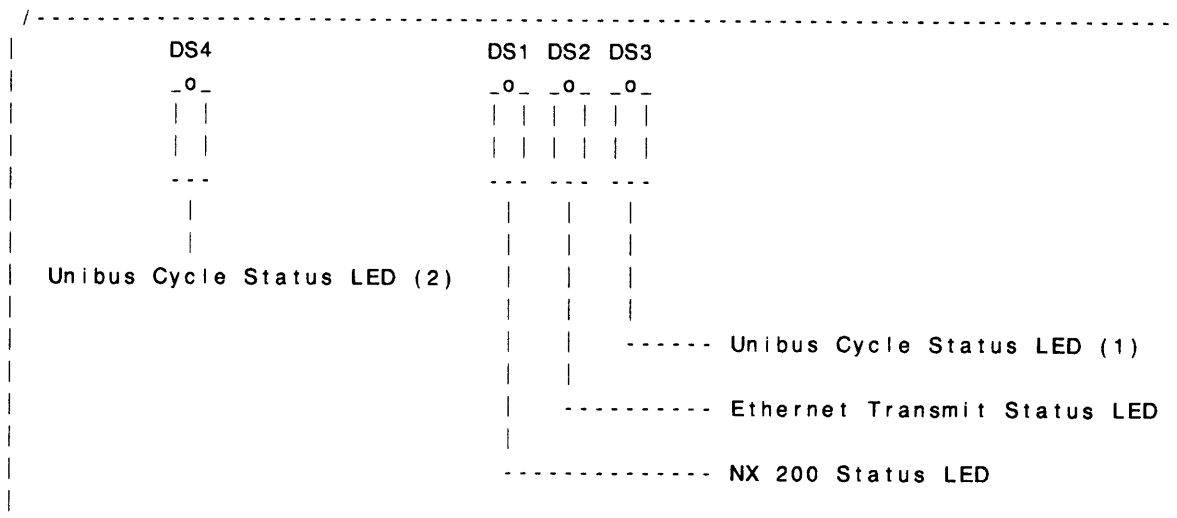


Figure 11-1: Quick Reference to Status LEDs

The Unibus addresses of I/O ports A and B are word addresses. For selection of an address, each of the 8 jumpers in J18 and each of the 3 jumpers in J16 must be appropriately selected. Address bits 0 and 1 are not selectable - they are always zero. Also, address bits 13 thru 17 are not selectable; they are always 1. As shipped from the factory, except J16-2, all jumpers are installed. Consequently the Unibus addresses 764000 and 764002 (Octal) are selected for I/O ports A and B respectively.

11.2.4. Unibus Interrupt Mechanism

The EXOS 204 can assert bus-vectored interrupts on the Unibus. Interrupt request priority level is jumper-selectable in the range from level 4 thru 7. Factory setting is for level 5.

The EXOS 204 can also be initialized to generate memory-mapped interrupts to the host. The host interrupts the EXOS 204 by writing to an I/O port (see Section 4.1).

11.2.5. Unibus Priority Resolution

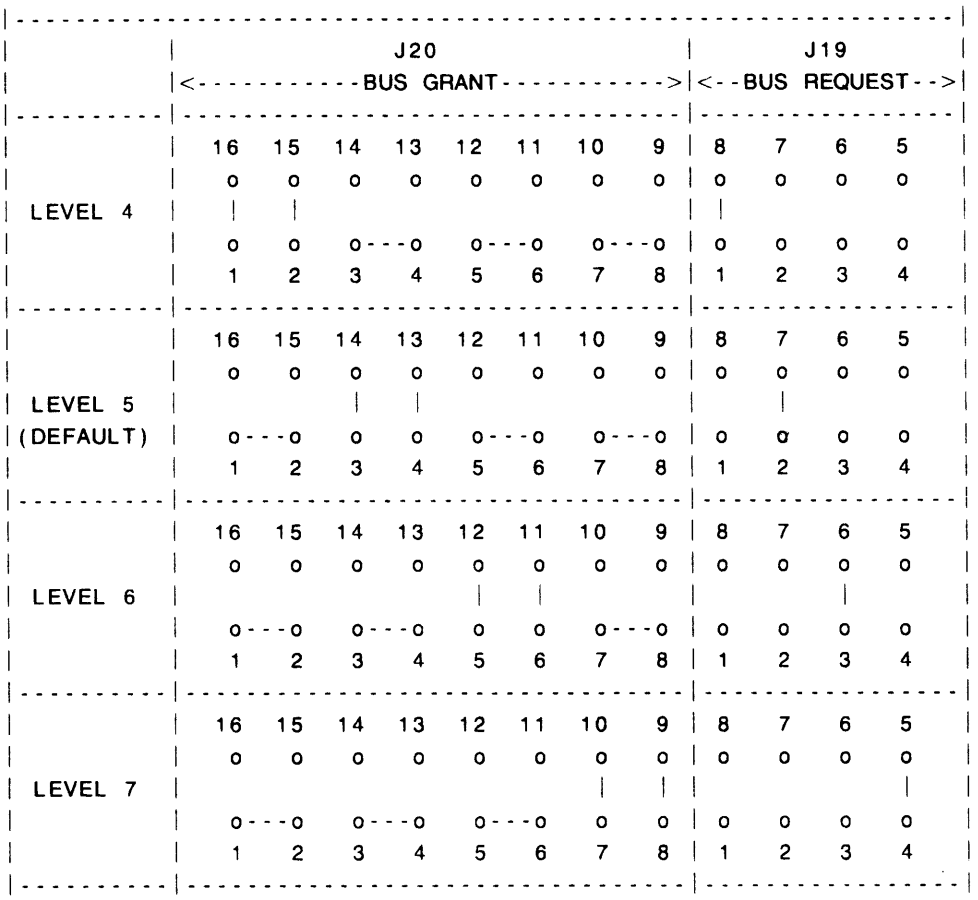
When several devices contend for bus mastership, the Unibus system grants control to the device physically closest to the processor module. Accordingly, the EXOS 204 should be installed closer to or farther from the processor module depending on the priority desired. The EXOS 204 uses the Unibus NPR line to transfer data to and from the host memory.

11.2.6. DMA Burst Length Selection

The EXOS 204 permits setting of DMA burst setting of 2, 4, 8, or 16 words per bus grant. Jumpers J17-1 and J17-2 set the DMA burst length as follows:

J17-1	J17-2	burst-length	
Installed	Installed	2	
Installed	Absent	4	(default setting)
Absent	Installed	8	
Absent	Absent	16	

Table 11-2: Interrupt Priority Set-Up Table



11.2.7. Unibus Error Interrupt Option

When Jumper J17-3 is present, the bus controller interrupts the 80186 if an error is detected on the Unibus; during the time the EXOS 204 is acting as the bus master (error is either bus timeout or parity error during read). When this jumper is not in and a bus error occurs, the bus hangs up, no interrupt is issued, and the LED DS4 lights up. (See also Section 11.2.8.)

11.2.8. Unibus Cycle Status LED

The Light Emitting Diodes (LEDs) in positions DS3 and DS4 on the EXOS 204 indicates that a Unibus cycle is in progress as follows. If DS4 is constantly lit, then the EXOS 204 has probably attempted to access a non-existent address on the Unibus. In general, this condition points toward a user software bug. Note that this applies only if the bus time-out feature is not selected by J17-3. If DS3 is constantly lit and DS4 is not lit, then most likely the EXOS 204 is unable to get DMA grant from the Unibus.

11.3. Ethernet Interface

Integrated with a standard Ethernet transceiver, the EXOS 204 performs all specified Ethernet physical and link level functions.

11.3.1. Ethernet Compliance

The EXOS 204 conforms fully to Ethernet specification, version 1.0, published September 30, 1980, and to Ethernet specifications version 2.0 published November, 1980, by DEC, Intel, and Xerox.

11.3.2. Ethernet Functions

Functions implemented on the EXOS 204 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.

11.3.3. Ethernet Address Recognition

The EXOS 204 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 204 also provides a promiscuous mode, in which it accepts all addresses.

Each EXOS 204 board has a unique 48-bit Ethernet address, stored in a PROM. 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 204 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 200 firmware control, the EXOS 204 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 1FFFFH in the EXOS 204'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 200 firmware, and transmitted with minimal delay.

11.3.6. Ethernet Error Handling

The EXOS 204 can be selectively enabled to receive packets normally rejected due to CRC and alignment errors.

11.3.7. Ethernet Transmit Status LED

The EXOS 204 lights an LED at position DS2 while transmitting on the Ethernet.

11.3.8. Ethernet Transceiver Connector

The EXOS 204 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 204 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 80186 CPU, clock speed 8 MHz.

128K of dual-ported RAM, 124K available for user software.

optionally, an additional 128K of dual-ported RAM.

NX 200 OS kernel, residing in two 16-Kbyte EPROMs.

Access to RAM is subject to wait states; net effective throughput is equivalent to an 80186 running at 5 MHz or faster, without wait states. Access to EPROM does not incur any wait states.

The NX 200 kernel provides a real-time, multi-tasking environment for the implementation of higher level protocols on the EXOS 204. It is supported by clock timer and interrupt controller chips. NX 200 implements consistent and portable access methods for the Ethernet and host interfaces. In addition, it executes self-diagnostics, and can optionally drive the EXOS 204 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 J7 selects NX 200 firmware options as follows:

> If J7 is installed, the EXOS 204 will attempt to down-load software from the Ethernet after self-test is complete. If not installed, the EXOS 204 will await initialization from the host after self-test is complete.

11.6. Self-Test Operation

When the EXOS 204 is reset by the Unibus INIT/ line or by host software (see Section 4.3), NX 200 firmware runs comprehensive diagnostic tests on EXOS 204 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 200 Status LED

Test progress and status are also reported via an LED at position DS1. On EXOS 204 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 204 is initialized by the host or from the Ethernet. After initialization, LED DS1 turns 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. Table 11-3 specifies all defined error codes for the EXOS 204.

Table 11-3: Self-Diagnostic and Configuration Error Codes

Hex Code	Pulse Code	Explanation of Error Code
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.
B0H	-.--	checksum on NX 200 EPROMs failed.
B1H	-.-- ...-	memory test failed for 0-128K.
B2H	-.-- ..-.	memory test failed for 128K up to the highest address.
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 PROM failed.
BAH	-.-- -.-.	system error.
BBH	-.-- -.--	Ethernet chip initialization failed.
BCH	-.-- --..	Ethernet chip self-test failed.
BDH	-.-- --.-	Ethernet chip resource counter failed.

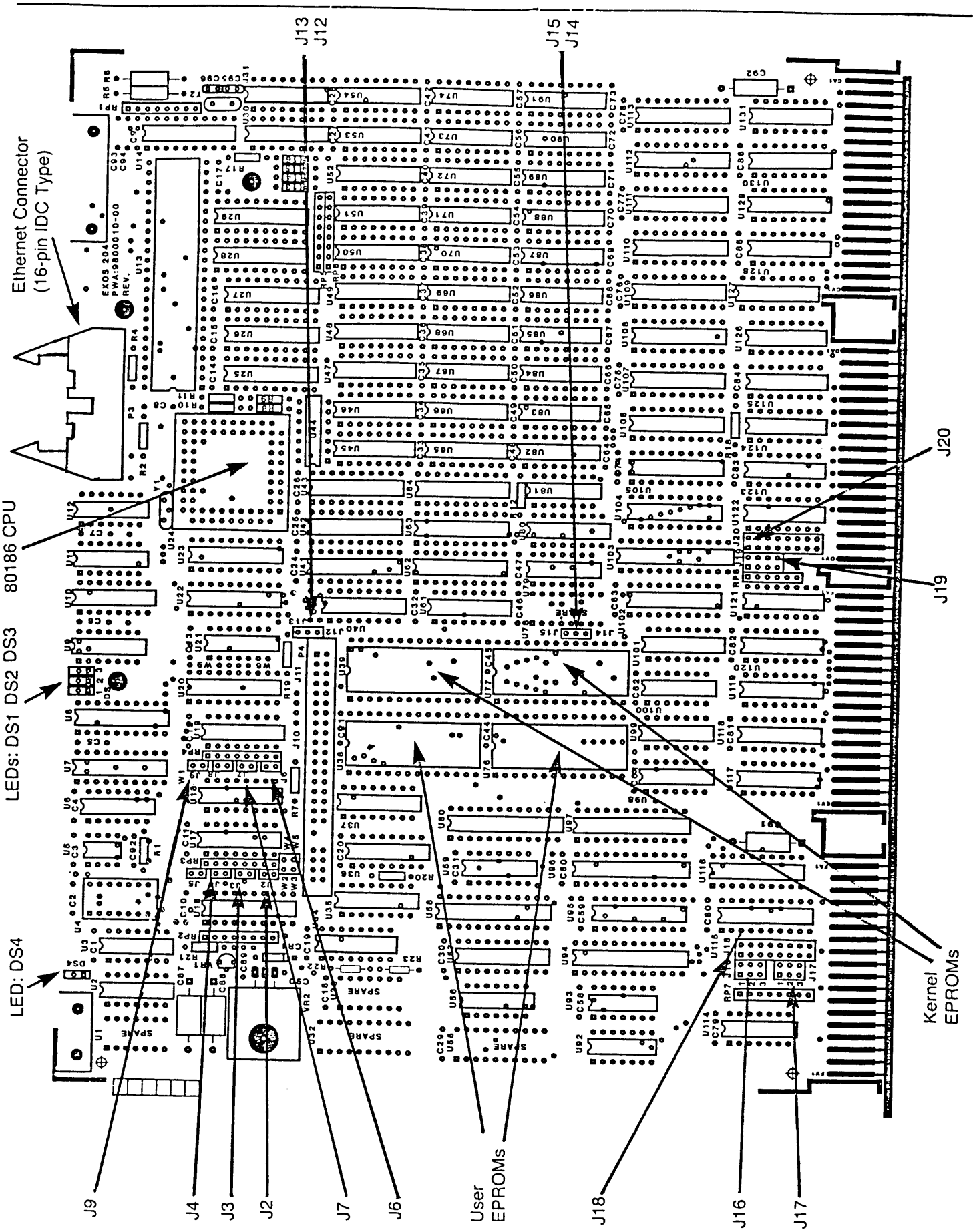
11.7. General Specifications

The following are general specifications for the EXOS 204.

Power Requirement:	+5VDC @ 5.0 Amp Max +15VDC @ 0.6 Amp Max (for transceiver and SBX connector) -15VDC @ 0.1 Amp Max (for SBX connector only)
Operating Environment:	Temperature: 5 to 50 degrees C Humidity: 0 to 90% without condensation
I/O Register Addresses:	Jumper-selectable (from 760000 to 777774 in increments of 2 (octal)). Factory setting: Port A : 764000, Port B : 764002
Interrupt Vector Address:	Software programmable.
Interrupt Priority Level:	Jumper selectable (BR4, BR5, BR6, or BR7).
Data Transfer:	Direct Memory Access (DMA) with jumper selectable burst size (2, 4, 8 or 16 word per NPR) for all word-aligned data.
Unibus Timeout:	16 micro-sec after MSYN asserted and no SSYN returning (jumper selectable).
Unibus Loading:	AC - 2 DEC unit loads DC - 1 DEC unit load
Physical Size:	1 6-layer DEC quad-sized PCB

(blank page)

APPENDIX A: COMPONENT LOCATION



(blank page)

EXCELAN

2180 Fortune Drive
San Jose, CA 95131
(408) 945-9526