THESIS

A LAYERED COMMUNICATION SYSTEM FOR ETHERNET

by

Mark D. Stotzer

September 1983

Thesis Advisor: U. R. Kodres

Approved for public release, distribution unlimited
A Layered Communication System for Ethernet

Approved for public release, distribution unlimited

Connecting heterogeneous computer systems via local area networks presents a challenge to software designers for the development of effective, reliable, and modifiable network communication software. This thesis presents a set of hierarchical program modules written for use on any INTELLEC MDS microcomputer development system, running the CP/M-80 operating system, to allow the...
system to become part of an Ethernet local area network. These program modules were written to not only obey the principles of software engineering, but to also reflect the same functional hierarchy as the International Standards Organization Open System Interconnection (ISO OSI) architectural reference model for computer networks.
A Layered Communication System for Ethernet

by

Mark D. Stotzer
Captain, United States Marine Corps
B.S., University of Louisville, 1977

Submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
September 1983

Author: 
Mark D. Stotzer

Approved by: 
Luo R. Kriens
Thesis Advisor

Mitchell B. Cotton
Second Reader

Robert D. Stone
Chairman, Department of Electrical Engineering

Dean of Science and Engineering
ABSTRACT

Connecting heterogenous computer systems via local area networks presents a challenge to software designers for the development of effective, reliable, and modifiable network communication software.

This thesis presents a set of hierarchical program modules written for use on any INTELLEC MDS microcomputer development system, running the CP/M-80 operating system, to allow the system to become part of an Ethernet local area network. These program modules were written to not only obey the principles of software engineering, but to also reflect the same functional hierarchy as the International Standards Organization Open System Interconnection (ISO OSI) architectural reference model for computer networks.
TABLE OF CONTENTS

I. INTRODUCTION------------------------------------------------- 13
  A. DISCLAIMER------------------------------------------------ 13
  B. BACKGROUND------------------------------------------------- 11
  C. PURPOSE--------------------------------------------------- 12

II. COMPUTER NETWORKS------------------------------------------- 14
  A. DEFINITION------------------------------------------------ 14
  B. PURPOSE--------------------------------------------------- 14
  C. THEORY---------------------------------------------------- 15
  D. LOCAL AREA NETWORKS--------------------------------------- 18
  E. ETHERNET-------------------------------------------------- 19

III. NETWORK IMPLEMENTATION------------------------------------- 21
  A. TOPOLOGY-------------------------------------------------- 21
  B. HARDWARE------------------------------------------------- 21
  C. SOFTWARE-------------------------------------------------- 26
     1. TEST PROGRAMS------------------------------------------- 26
     2. COMMUNICATION BETWEEN NETWORK HOSTS------------------ 27
  D. OPERATION------------------------------------------------- 32
  E. PERFORMANCE--------------------------------------------- 33

IV. CONCLUSIONS------------------------------------------------- 36

APPENDIX A: NI3010 COMMAND LISTING--------------------------- 38
APPENDIX B: NI3010 REGISTER LISTING-------------------------- 39
APPENDIX C: NI3012 STATUS REGISTER CODES-------------------- 40
APPENDIX D: TRANSMIT DATA FORMAT 41
APPENDIX E: RECEIVE DATA FORMAT 42
APPENDIX F: SOURCE CODE OF PROGRAM ETHTESTA.ASM 43
APPENDIX G: SOURCE CODE OF PROGRAM ETHTESTB.ASM 52
APPENDIX H: SOURCE CODE OF MAIN MODULE ETHERNET.PLL 61
APPENDIX I: SOURCE CODE FOR MODULE SENDATA.PLL 69
APPENDIX J: SOURCE CODE FOR MODULE RECDATA.PLL 73
APPENDIX K: SOURCE CODE FOR MODULE ETHER2.ASM 76
APPENDIX L: TEST PROGRAM USER INSTRUCTIONS 91
APPENDIX M: COMMUNICATION PROGRAM USER INSTRUCTIONS 92
LIST OF REFERENCES 95
BIBLIOGRAPHY 96
INITIAL DISTRIBUTION LIST 97
<table>
<thead>
<tr>
<th>Table</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Type Field Protocol</td>
<td>29</td>
</tr>
<tr>
<td>3.2</td>
<td>Comparison of Program Modules and ISO Model</td>
<td>31</td>
</tr>
<tr>
<td>3.3</td>
<td>Performance Comparison for Data Transfers Between Single and Double Density INTELLEC Systems</td>
<td>33</td>
</tr>
<tr>
<td>3.4</td>
<td>Performance Comparison of Transfers Between VAX 11/780 and INTELLEC Systems</td>
<td>34</td>
</tr>
<tr>
<td>3.5</td>
<td>Maximum Performance Data</td>
<td>35</td>
</tr>
</tbody>
</table>
LIST OF FIGURES

2.1 ISO Reference Model--------------------------- 17
3.1 N13010 Switch and Jumper Locations--------------- 22
3.2 Ethernet Architecture and N13010 Implementation---- 23
ACKNOWLEDGEMENTS

To my wife, Jan, goes special thanks for her unlimited patience and for our new son, Mark Andrew.

Additionally, an expression of thanks to Mr. Mike Williams, Computer Science Professional Staff, for his expert advice and assistance throughout this project.
I. INTRODUCTION

A. DISCLAIMER

Many terms used in this thesis are registered trademarks of commercial products. Rather than attempt to cite each individual occurrence of a trademark, all registered trademarks appearing in this thesis are listed below following the firm holding the trademark:

Digital Research Incorporated, Pacific Grove, California
- CP/M-80 Operating System
- CP/M-86 Operating System
- PL/I-80 Programming Language
- PL/I-86 Programming Language
- LINK-80 Linking Utility
- XLT-86 Code Conversion Utility

Intel Corporation, Santa Clara, California
- INTELLEC MDS Microcomputer Development System
- Multibus Bus Architecture
- 8080/8086 Microprocessors
- 8080 Assembly Language Programming Language
- ISIS-II Operating System
- IAPX-432 Development System

Digital Equipment Corporation, Maynard, Massachusetts
- VAX 11/790 Minicomputer
VAI/VMS Operating System
Interlan Corporation, Chelmsford, Massachusetts

N13010 Ethernet Controller Board
Xerox Corporation, Stamford, Connecticut

Ethernet Local Area Network

B. BACKGROUND

The connection of heterogeneous computer systems via some form of network, to perform various data processing tasks where data or resource sharing is important, is an extremely active topic for both hardware and software designers.

The International Standards Organization Open System Interconnection (ISO OSI) architectural reference model provides the general framework in which computer network systems are designed to operate. This seven-layered, hierarchical description of functions was developed to provide a vehicle for the later development of a set of specific network protocols. The hierarchical nature of this model compares favorably with the techniques of hierarchical, structured design of software that are being taught and implemented today. The logical conclusion of the above comparison is to use the functionally layered framework provided by the ISO OSI model as a guide for deciding how to modularize the communication software necessary to allow host computers to be connected via a network.
C. PURPOSE

The main purpose of this thesis is to construct a software interface to the CP/M-80 operating system so that files and messages can be transported between various host systems via a Local Area Network. The structuring of this software, to reflect the layers of the ISO model, allows modifications to the network software to be more easily made.

This thesis presents a set of PL/I and Intel 8080 Assembly Language modules that, when linked together, allow INTELLEC MDS users to communicate via an Ethernet Local Area Network. The complete set of software developed also includes two programs that can be used to troubleshoot or test the Ethernet hardware. The communication program allows INTELLEC MDS computers connected to the network to:

1. Send messages or files to other hosts.
2. Receive messages or files from other hosts.
4. Command file transfers to or from the VAX.

Additionally, the communication software will provide faster data transfers between host machines than the direct host-to-host serial communications methods currently used.

This thesis is divided into four chapters. Chapter II discusses computer networks in general. The Ethernet is presented as a specific example of a Local Area Network. The Interlan hardware is also discussed as an implementation of the Ethernet. Chapter III deals with the details of the
Ethernet communications software. The topological, hardware, software and performance issues are presented in detail. Chapter IV presents the conclusions drawn from the network realization and discusses possible areas of future growth and performance enhancement.
II. COMPUTER NETWORKS

A. DEFINITION

Computer networks are defined to be collections of interconnected, autonomous computers. A computer network can also be a grouping in which the required processing functions are dispersed among several of the attached hosts. [Ref. 1: p. 2]

Computer networks are classified by their length. Networks whose attached hosts are farther than a few kilometers apart are considered Long Haul, while shorter networks are considered Local Area. Networks are also classified by the nature of the hosts connected to them. Homogeneous networks consist of like hosts, while heterogeneous networks consist of dissimilar hosts.

B. PURPOSE

The main reason that the subject of computer networking has rapidly achieved prominence is that networking provides a workable solution to data processing problems where the sharing of data or other resources is important. Networking can also enhance the fault tolerance of an activity's computational assets. Loss of any host, connected to most Local Area networks, would not affect either the other hosts or the network itself. [Ref. 1: pp. 3-4]
Current trends seem to point to the merging of personal computers with Local Networking to form what one author calls "community microcomputing" [Ref. 2: p. 60]. This refers to the interconnection, via a Local Area Network, of a set of microcomputers that may, as a networked group, enhance the price/performance ratio for the using activity when compared to installing a single, large mainframe computer [Ref. 1: p. 5].

C. THEORY

The most generally accepted model of computer network architecture is the International Standards Organization Open Systems Interconnection Model (ISO OSI) model. This model is a set of hierarchical functions and protocols that are necessary to allow computers to communicate via a network. The seven layers and their definitions are listed below: [Ref. 1: pp. 15-21]

1. Physical Layer – This layer provides the actual connection between hosts. It provides the bit stream transmission across the network medium.

2. Data Link Layer – This layer performs error detection and correction, address recognition and flow control. This layer also provides data framing if necessary.

3. Network Layer – The network layer provides logical channels between two endpoints in a network. This layer forms the data into packets for transmission.

4. Transport Layer – The transport layer provides the network with single, group, or broadcast addressing modes and sets up virtual circuits between hosts.

5. Session Layer – This layer contains the functions necessary to perform address conversion. This layer
initiates, binds, and terminates the dialogue between hosts.

6. Presentation Layer — The presentation layer is mainly concerned with converting and transforming the data passed to a user. This layer also contains the file transfer and virtual protocols.

7. Application Layer — The application layer, the highest in the model, is where the user interface to all the network services resides. The lower layers exist only to support this layer.

Many computer networks with layered protocols exist, but their layers may not match the ISO model exactly because some of the ISO functions may not be necessary. The development of the model came about due to the need to standardize network description. The main factors that motivated the designers were: [Ref. 1: p. 15]

1. To create a layer where abstraction was necessary.
2. To give each layer a well defined function.
3. To keep the information passed between layers to a minimum.
4. To create only a minimum number of layers to decrease complexity.

The above design principles are the same as the software engineering principles of abstraction and modularity. The hierarchical structure also compares favorably with the structured programming techniques of software design that are currently being advocated. [Ref. 4: pp. 58-60]

The ISO OSI model is shown in Figure 2.1. The main concepts of the model are: [Ref. 8: pp. 28-29]

1. Each layer only interacts with the vertically adjacent layers through well defined interfaces.
Changes to any layer can thus be accomplished without changing the other layers.

2. Two basic protocols exist per layer. The first is the vertical protocol between layers. The second is the horizontal or peer protocol between transmitting and receiving layers of different hosts that allows virtual communication to occur between those hosts.

![ISO Reference Model](image)

The flow of data in the network model begins at the top layer of the sending host. As the data is passed down the sending host’s layers additional information, either bits or bytes, is added to the original data until the lowest layer...
is reached. At the lowest layer, the data and added information is sent on the network medium. The receiving host then performs the reverse process on the received information by passing it up the ISO layers until all that remains, after again reaching the top layer, is the original data.

D. LOCAL AREA NETWORKS

Computer networks, as previously mentioned, are classified as either Long Haul or Local Area. Local Area networks are characterized by: [Ref. 1: p. 286]

1. A length of no greater than a few kilometers.
2. A data rate in excess of one million bits per second (1 Mbps).
3. Ownership by a single organization.

Two techniques of transmission medium access are being considered for standardization by the Institute of Electrical and Electronic Engineers (IEEE). The proposed IEEE Standard 802 endorses both the token passing and carrier sense methods of Local Area Network medium access. Token passing consists of not allowing any host on the local network to transmit on the medium unless it has possession of a token that is passed in a predetermined order from one host to another. The carrier sense method allows each host equal access to the network. This scheme allows each host to detect the occurrence of any other transmissions on the network and allows the host to wait until the medium is
clear before transmitting. If two hosts try to transmit simultaneously, they will each detect the collision and wait an independent, random interval before attempting another transmission. Ethernet is an example of a carrier sense network. [Ref. 5: p. 31]

E. ETHERNET

Specific details of Ethernet Standard – Version 1.0 are:

[Ref. 6: p. 1]

1. A data rate of 10 Megabits per second (10Mbps).
2. A maximum host separation of 2.5 kilometers.
3. A transmission medium consisting of a shielded coaxial cable.
4. A topology consisting of an unrooted tree.
5. Link control via fully distributed peer protocol with statistical contention resolution.
6. A message protocol of variable size frames.

Additionally, it must be noted that the Ethernet Standard does not provide for either error correction, data encryption, or priority access to the network medium. At any point in time, only one transmission can occupy the medium. [Ref. 6: p. 5]

One current implementation of an Ethernet network is the E-BUS system developed by E-Systems Incorporated. The E-BUS implementation differs from the Ethernet Standard in that it provides for transmitted frames to be acknowledged. The E-BUS also provides multiple coaxial cables to increase both
the effective bandwidth and the overall fault tolerance of
the network. [Ref. 10: pp. 77-78]
A. TOPOLOGY

The Ethernet Local Area Network implemented at the Computer Science Department of the Naval Postgraduate School consists of three connected systems:

1. The VAX 11/780 (VMS operating system) minicomputer.

2. An INTELLEC MDS system (CP/M-80 operating system), with attached double density disk drives, that functions as the input/output processor for the Intel INAP 432 32 bit microcomputer system.

3. A second INTELLEC MDS system with attached single density disk drives. (Also CP/M-80)

This thesis presents the software necessary to allow the above CP/M-80 based systems to communicate via the network. The software necessary to allow the VAX 11/780 the same communication capabilities was written by Lt. Thawip Netniyom [Ref. 9].

B. HARDWARE

All the hardware needed to implement the above network was provided by the Interlan Corporation. The hardware needed to connect each INTELLEC system to the network was installed as follows: [Ref. 7: pp. 7-13]

1. The base port address switches and the priority and interrupt jumpers were set on the NI3010 Ethernet controller board as shown in Figure 3.1.

2. The NI3010 was then inserted into the INTELLEC system in an odd-numbered slot in the Multibus.
The NT10 transceiver was installed across the Ethernet coaxial cable and the cabling that connects the NT10 to the N13010 was connected as shown in Figure 3.2.

The above mentioned hardware provides the ISO layer one and two functions. The Physical Layer functions provided by the transceivers and connecting cables are: [Ref. 7: p. 2]

1. Support of a 10 Mbps data rate.
2. Bit stream generation through Manchester encoding.
3. Media access control.

Figure 3.1 N13010 Switch and Jumper Locations

The Data Link Layer functions provided by the N13010 board are: [Ref. 7: p. 2]

1. Data encapsulation/decapsulation (framing).
2. Address recognition.
3. Transmit and receive data link management.
The N13010 operates both as a slave to the host computer and as a master processor when controlling the direct memory access (DMA) operations between the N13010 buffers and the host computer's memory. The transmit function is command driven by the host, while the receive function is interrupt driven. Control of the N13010 by the host is accomplished by programming the host to load commands, addresses, byte counts and interrupt enable values into registers onboard the N13010. [Ref. 7: pp. 69-75]

![Figure 3.2 Ethernet Architecture and N13010 Implementation](image-url)
Appendix B. After issuance of any command, the host must check for a value in the Command Status Register. The execution of the command only occurs after this read operation has been accomplished. The details of the read operation are as follows: [Ref. 7: pp. 70-72]

1. The host issues a command.

2. The host checks the Interrupt Status Register to check if the least significant bit is a one. If the least significant bit is a one, then the host reads the value in the Command Status Register.

3. If the value in the Command Status Register is a zero then the command executed successfully. After the host has issued a Load, Transmit, and Send command, a value of one is also considered a success. Any other value represents a failure. A listing of Command Status Register values is located in Appendix C.

The Command Status Register must also be read at the beginning of any program written to control the NI3010. This register must be read at this time because the NI3010 automatically performs its built-in diagnostic routines each time the board is powered up or reset. The automatic testing places a value in the Command Status Register that must be read to clear the register before any other commands can be given to the NI3010.

The NI3010 transmit function is accomplished in the following manner: [Ref. 7: p. 85]

1. The host loads a block of memory in the format shown in Appendix D for each frame to be transmitted.

2. The host loads the three NI3010 address registers with the first address of the host memory block.
3. The host then loads the two NI3010 byte count registers with the number of bytes in the data block.

4. The host then enables a Transmit DMA Done (TDD) interrupt by writing a value of 6 Hex into the Interrupt Enable Register.

5. The NI3010 interrupts the host once the memory block has been transferred into the NI3010 transmit buffer.

6. The host then enables a Receive Block Available (RBA) interrupt by loading the Interrupt Enable Register with a value of 4 Hex. This step allows any pending received frames to be handled.

7. The host then commands the NI3010 to send the frame by writing a value of 29 Hex into the Command Register and subsequently reading the Command Status Register as previously discussed.

The NI3010 receive function is accomplished as shown below: [Ref. 7: p.90]

1. The host enables an RBA interrupt as shown above.

2. The NI3010, upon receiving a frame, interrupts the host to notify it of frame receipt.

3. The host then writes a value of 0 Hex into the Interrupt Enable Register to disable any other NI3010 interrupts.

4. The host writes values into the three NI3010 address registers to inform the NI3010 where, in host memory, to transfer the data.

5. The host then loads the two NI3010 byte count registers.

6. The host then enables the DMA transfer of the data by writing a value of 7 Hex into the Interrupt Enable Register.

7. The NI3010 then interrupts the host upon completion of the transfer. The format of received data in the host memory is shown in Appendix E.

The above steps are repeated for each received frame. The host is then responsible for whatever further operations...
must be done with the data. For example, the data could be displayed on the console or written to a disk file.

The N13010 also has built-in test features and can also support the concepts of broadcast and multicast transmission. Broadcast transmission allows a host to transmit to all other hosts simultaneously, while multicast allows transmission to only a few selected hosts.

C. SOFTWARE

The software necessary to implement ISO layers three through seven was originally written entirely in 8080 Assembly Language. The final version of the communication program consists of PL/I-80 modules that perform the functions of ISO layers six and seven and an Intel 8086 Assembly Language module that performs the functions of ISO layers two and three. The ISO layer two functions performed by the software supplement the functions of this layer performed by the N13010. The primary goals of the software were:

1. To allow users to run, if necessary, test programs that will verify the functioning of the hardware.

2. To allow the INTELLEC systems to communicate via the Ethernet to any other hosts connected to the network.

1. Test Programs

The basic software design process began by first determining the major functional divisions or modules into which a program should be divided. A primary consideration,
since implementation using the N13010 is interrupt dependent, was a simple interrupt handling routine. This routine was the basis of the first working test program, ETHTESTA. The interrupt handling module is the basis around which all the succeeding programs were written. ETHTESTA, an 8080 Assembly Language program, commands the N13010 to perform built-in tests, one of which sends test data to the N13010 Transmit buffer and back through the N13010 Receive Data Register. This process is called the N13010 Module Interface Loopback mode. Use of this test mode does not permit the interrupt handling to be done in the same manner as a normal communication program, nor does this mode allow data to be sent onto the network. The source code listing of ETHTESTA.ASM is located in Appendix F.

A process of gradual enhancement was then applied to upgrade ETHTESTA into a program that utilized the complete interrupt capability as that of a functional communication program. The follow-on test program, ETHTESTB, performs all the tests of ETHTESTA and, additionally, sends a small block of data to itself via the network using the N13010 Internal Loopback mode. A source code listing of ETHTESTB.ASM can be found in Appendix G.

2. Communication Between Network Hosts

The test programs discussed previously involved the utilization of only one INTELLEC system with installed Ethernet hardware. The next logical step was to again
upgrade the software to allow the INTELLEC systems to communicate via the network.

In order to give hosts, especially of different architectures and operating systems, the ability to communicate via a network involves the development of higher level protocols to handle any differences that may arise due to the above factors. Specifically, differences between hosts related to file storage and frame transmission speed are the kind of issues that must be handled by the use of protocols. In an Ethernet network, the nature of each frame sent onto the network must also be encoded so that the receiving host can determine what further operations must be performed on the received frame data.

The primary operating system file storage mismatch in this network implementation occurred between the VAX/VMS and the CP/M-80 operating systems. The VAX stores text files as variable length records by text sentence. The VAX, also, does not explicitly store the carriage return and line feed characters in the record. On the other hand, the CP/M-80 operating system stores all the characters, including the carriage return and line feed, in one long continuous file. This file storage incompatibility was resolved by adding format conversion routines to both the VAX and INTELLEC software to convert the data prior to transmission on the network.
A transmission versus reception speed mismatch was discovered in the early testing between the VAX and the INTELLEC systems. The VAX can send data much faster than the INTELLEC systems can receive it. The solution to this problem was to add a "stop-and-wait" [Ref. 1: pp. 143-145] protocol to the ISO layer two functions already performed by the NI3010. This protocol was implemented in software and assures the sending host that the last frame sent was correctly received. This protocol also prevents a faster sender from inundating a slower receiver.

The frame encoding protocol adopted for our network is as shown in Table 3.1. These codes are written into the two Type Field bytes in the transmit data block as shown in Appendix D. The receiving hosts interpret these two bytes, once the data block is in their memory as shown in Appendix E, to determine what operations must be done to the data.

<table>
<thead>
<tr>
<th>Type Field</th>
<th>Interpretation at Receiver</th>
</tr>
</thead>
<tbody>
<tr>
<td>$00F$</td>
<td>Message frame</td>
</tr>
<tr>
<td>$00$</td>
<td>Last frame of terminal reply</td>
</tr>
<tr>
<td>$00$</td>
<td>Acknowledge frame</td>
</tr>
<tr>
<td>$01$</td>
<td>File transfer-middle frame</td>
</tr>
<tr>
<td>$0F$</td>
<td>File transfer-first frame</td>
</tr>
<tr>
<td>$0F$</td>
<td>File transfer-last frame</td>
</tr>
</tbody>
</table>

Table 3.1 Type Field Protocol
The other protocol adopted was to use fixed data block sizes per Ethernet frame. The choices available to the user are:

1. 128 Bytes. (Must be used for all file transfers)
2. 256 Bytes.
3. 512 Bytes.
4. 1024 Bytes.
5. 1500 Bytes. (Used in VAX terminal service mode)

A set of programs, written exclusively in 8080 Assembly Language, was first developed to send short, single sentence messages from one INTELLEC system to another using the above protocols. Next, the file transfer modules were developed and tested. Throughout the entire process, close attention was paid to maintaining software modularity that was analogous to the functional modularity of the ISO model. Software modules that compared directly to ISO layers were maintained as separate modules and, whenever possible, rewritten in PL/I-80, a high level language. The final communication program consists of three PL/I-80 modules and one 8080 Assembly Language module. These modules were linked together, using LINK-80, into the final product. The final program, ETHERNET.COM, contains calling sequences that directly reflect the ISO OSI model structure as shown in Table 3.2. The source code for all modules can be found in Appendices H through K. Modules were not written for ISO layers four and five because these layers are primarily
concerned with Long Haul network functions that are unneeded by our network. Modules RECEIVE, SENDFRAM, RECFRAM, TRMSG and AWAIT are contained in the assembly language module because the functions they are required to perform are more efficiently programmed in that language. The actual calling sequence for the transmit process occurs as follows:

1. ETHERNET: Asks for user to select type of network service desired and calls SENDATA.

2. SENDATA: Encodes the transmit type field for the user selected service and calls internal routines to control the transmission. This module calls SENDFRAM as each frame is ready for sending.

3. SENDFRAM: This module sends each frame onto the network then calls AWAIT to wait for the acknowledge frame to arrive from the destination host.

Table 3.2 Comparison of Program Modules and the ISO Model

<table>
<thead>
<tr>
<th>ISO LAYER</th>
<th>File</th>
<th>Transmit Message</th>
<th>VAX Modes</th>
<th>Receive File</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>ETHERNET.PL</td>
<td>Same</td>
<td></td>
<td>RECEIVE(ETHER2.ASM)</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>SENDATA.PL</td>
<td>Same</td>
<td></td>
<td>RECDATA.PL</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Not Implemented</td>
<td>Same</td>
<td></td>
<td>Not Implemented</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Not Implemented</td>
<td>Same</td>
<td></td>
<td>Not Implemented</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>SENDFRAM(ETHER2)</td>
<td>Same</td>
<td></td>
<td>RECFRAM(ETHER2)</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>AWAIT(ETHER2)/Hdwe</td>
<td>Same</td>
<td></td>
<td>TRMSG(ETHER2)/Hdwe</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>NT10 Hardware</td>
<td>Same</td>
<td></td>
<td>Same</td>
<td>Same</td>
</tr>
</tbody>
</table>

The calling sequence for the receive process is in the order shown below:
ETHERNET: The user selects the receive mode of network service and this module calls RECEIVE.

RECEIVE: This module waits in a loop for the module RECFRAM to receive a frame from the network. Once the receive data is placed in host memory by RECFRAM, a flag is set and RECEIVE calls RECDATA.

RECDATA: This module decodes the type field of the received frame and calls internal modules that handle each different type of received data and, as part of this process, calls TRMSG which send the acknowledge frame back to the source.

The four major functions that the final program performs are:

1. Transmission of files or messages to any other network hosts.
2. Reception of files or messages from any other hosts.
3. The ability to become a terminal of the VAX 11/782 via the Ethernet.
4. The ability to send specially coded messages to the VAX to command it to either upload or download files.

D. OPERATION

The operation of test programs, ETHTESTA and ETHTESTB, consists primarily of invoking either program using normal CP/M-80 procedures and following the directions presented by the program. Detailed instructions for use of the test programs can be found in Appendix L.

Operation of the communication program, ETHERNET, also involves invoking the program using normal CP/M-80 procedures and following the menus presented by the program. Detailed operating instructions for the use of the final communication program are located in Appendix M.

32
E. PERFORMANCE

The communication program provides faster data transfer between network hosts than currently employed methods. Table 3.3 demonstrates the improved performance realized in transferring data between single and double density INTELLEC systems.

|TABLE 3.3 Performance Comparison for Data Transfers Between Single and Double Density INTELLEC Systems |
|---|---|---|---|---|
|**Software Used** | **File Size (KBbytes)** | **Time (Min:Sec)** | **Data Rate (bps)** | **Effective (%)** |
|SDKFER | 136 | 22:45 | 9600 | 797 |
|ETHERNET | 136 | 3:30 | 10M | 518 |

The data rate of the medium is the rate at which data is actually sent on whatever medium is being utilized. The effective data rate is the number of bits of useful data that was sent divided by the total elapsed time of the data transfer. Data transfers between INTELLEC systems were not the only ones that showed a significant improvement over methods that were previously utilized. Transfers of data to and from the VAX 11/780 were also accomplished significantly faster as shown in Table 3.4.

The below presented data shows the improved performance of data transfers when the Ethernet network is employed. Lastly, a series of experiments was performed to investigate
the performance limits of data transmission and reception of the CP/M-80 based programs. The conditions of the

Table 3.4 Performance Comparison of Transfers Between VMX 11/780 and INTELLEC Systems

<table>
<thead>
<tr>
<th>Software Utilized</th>
<th>File Size</th>
<th>Time (Min:Sec)</th>
<th>Data Rate (bps)</th>
<th>Medium</th>
<th>Effective</th>
</tr>
</thead>
<tbody>
<tr>
<td>IAPX 432 Pkg</td>
<td>136</td>
<td>6:40</td>
<td>9600</td>
<td></td>
<td>2720</td>
</tr>
<tr>
<td>ETHERNET (To disk file)</td>
<td>136</td>
<td>2:05</td>
<td>10M</td>
<td>6704</td>
<td></td>
</tr>
<tr>
<td>ETHERNET (To memory buffer)</td>
<td>136</td>
<td>1:35</td>
<td>10M</td>
<td>11452</td>
<td></td>
</tr>
</tbody>
</table>

experiments were:

1. The stop-and-wait protocol was not employed.
2. The frames would be sent as fast as possible using the minimum amount of 8080 Assembly Language code.
3. The receiver would not perform any extra operations on received data other than that done by the 113010. No data was either written to any disk files or displayed on the console.
4. Testing was done on data block sizes of 128 and 1500 bytes per ETRENIT frame.

Testing was performed between two INTELLEC systems and data was collected for both the above data block sizes. The results of the experiments are shown in Table 3.5.

As shown below, the highest data rate achieved was 1.764 Megabits per second. The time taken in each 6.8 millisecond period was accounted for as follows:

1.2 msec  Actual Data Transmission of 1500 Bytes

34
0.5 msec Instruction Execution to Restart Transmit
3.5 msec DMA Operation of 1500 Bytes at 428 KBps rate
1.6 msec Execution Time of NI3010 Send Command

6.8 milliseconds total

Table 3.5 Maximum Performance Data

<table>
<thead>
<tr>
<th>Data Bytes per Frame</th>
<th>Frame Transmission Interval</th>
<th>Data Rate (Effective)</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>2.7 Milliseconds</td>
<td>379 Kbps</td>
</tr>
<tr>
<td>1500</td>
<td>6.8 Milliseconds</td>
<td>1.764 Mbps</td>
</tr>
</tbody>
</table>

The conclusions reached about the Ethernet performance were:

1. The transmission speed is limited by the NI3010 controller itself. The NI3010 Send command required longer to execute than either the actual transmission time of the data or the instruction execution during each transmit cycle.

2. Although the NI3010 literature claims a DMA data rate of 1 MBps, the board could only achieve a rate of 428 KBps. This limitation could be due to the method in which the NI3010 onboard microprocessor is utilized.
IV. CONCLUSIONS

This thesis has shown that functional Local Area Network communication software can be structured according to the ISO OSI network model. This thesis has also shown that the performance of the Ethernet substantially reduces the transfer time of data between connected hosts when compared to methods previously employed. The single to double density transfer rate improved by a factor of 7.5 while the VAX to INTELLEC transfer rate improved by a factor of 3.2. The data also shows that effective data rates can be improved by faster host processors, but that hosts will be limited by the rate at which the N13010 can transfer data to and from host memory and then send it. INTELLEC hosts are also limited in actual network use by the rate at which data can written to or read from disk drives.

An improvement to the effective data transmission rate might be realized by synchronizing the speed between sending and receiving hosts by some method other than the stop-and-wait protocol utilized in this thesis. The transmission rate performance degradation noted above is only aggravated by using the stop-and-wait protocol.

The software written for this thesis can be adapted to run on an Intel 8086 based system by following the steps listed below:
1. The PL/I-90 source code files can be directly compiled using the PL/I-86 compiler.

2. The 8080 Assembly Language source code can either be hand-translated or translated by software such as the program XLT-86 into 8086 Assembly Language source code. It should be noted that there are differences between the 8080 and 8086 processors that have to do with how interrupts are handled that will require some rewriting of the converted code.
APPENDIX A
NI3010 COMMAND LISTING

<table>
<thead>
<tr>
<th>Code (Hex)</th>
<th>Command Function</th>
<th>Returned Code (Hex)</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>Set Module Interface Loopback</td>
<td>00</td>
</tr>
<tr>
<td>02</td>
<td>Set Internal Loopback</td>
<td>00</td>
</tr>
<tr>
<td>03</td>
<td>Clear Loopback</td>
<td>00</td>
</tr>
<tr>
<td>04</td>
<td>Set Promiscuous Mode</td>
<td>00</td>
</tr>
<tr>
<td>05</td>
<td>Clear Promiscuous Mode</td>
<td>00</td>
</tr>
<tr>
<td>06</td>
<td>Set Receive on Error Mode</td>
<td>00</td>
</tr>
<tr>
<td>07</td>
<td>Clear Receive on Error Mode</td>
<td>00</td>
</tr>
<tr>
<td>08</td>
<td>Go Offline</td>
<td>00</td>
</tr>
<tr>
<td>09</td>
<td>Go Online</td>
<td>00</td>
</tr>
<tr>
<td>0A</td>
<td>Run Onboard Diagnostics</td>
<td>Diagnostic Codes as shown in Appendix C</td>
</tr>
<tr>
<td>18</td>
<td>Report/Reset Statistics</td>
<td>00</td>
</tr>
<tr>
<td>19</td>
<td>Report Collision Delays</td>
<td>00</td>
</tr>
<tr>
<td>28</td>
<td>Load Transmit Data</td>
<td>00, 05</td>
</tr>
<tr>
<td>29</td>
<td>Load/Transmit/Send Data</td>
<td>00, 01, 03, 05, 06, 08, 0B</td>
</tr>
<tr>
<td>2A</td>
<td>Load Group Addresses</td>
<td>00, 05, 0A</td>
</tr>
<tr>
<td>2B</td>
<td>Delete Group Addresses</td>
<td>00, 05, 0A</td>
</tr>
<tr>
<td>3F</td>
<td>Reset</td>
<td>00</td>
</tr>
</tbody>
</table>

Notes: Promiscuous Mode receives all network traffic.
Receive on Error receives even bad frames.
## APPENDIX B

### NI3010 REGISTER LISTING

<table>
<thead>
<tr>
<th>Register Description</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command Status</td>
<td>Base Port Address+ 01H</td>
</tr>
<tr>
<td>Transmit Data</td>
<td>Base Port Address+ 02H</td>
</tr>
<tr>
<td>Receive Data</td>
<td>Base Port Address+ 03H</td>
</tr>
<tr>
<td>Status (Interrupt)</td>
<td>Base Port Address+ 05H</td>
</tr>
<tr>
<td>Interrupt Enable</td>
<td>Base Port Address+ 08H</td>
</tr>
<tr>
<td>Extended Bus Address</td>
<td>Base Port Address+ 09H</td>
</tr>
<tr>
<td>High Bus Address</td>
<td>Base Port Address+ 0AH</td>
</tr>
<tr>
<td>Low Bus Address</td>
<td>Base Port Address+ 0BH</td>
</tr>
<tr>
<td>High Byte Count</td>
<td>Base Port Address+ 0CH</td>
</tr>
<tr>
<td>Low Bus Address</td>
<td>Base Port Address+ 0DH</td>
</tr>
</tbody>
</table>

Note: The base port address is set on the DIP switch onboard the NI3010.
APPENDIX C
NI3010 STATUS REGISTER CODES

1. Normal Mode:

<table>
<thead>
<tr>
<th>Code (Hex)</th>
<th>Command Status Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Success</td>
</tr>
<tr>
<td>01</td>
<td>Success with Retries</td>
</tr>
<tr>
<td>02</td>
<td>Illegal Command</td>
</tr>
<tr>
<td>03</td>
<td>Inappropriate Command</td>
</tr>
<tr>
<td>04</td>
<td>Failure</td>
</tr>
<tr>
<td>05</td>
<td>Buffer Too Large</td>
</tr>
<tr>
<td>06</td>
<td>Frame Too Small</td>
</tr>
<tr>
<td>08</td>
<td>Excessive Collisions</td>
</tr>
<tr>
<td>0A</td>
<td>Buffer Alignment Error</td>
</tr>
</tbody>
</table>

2. Diagnostic Mode:

<table>
<thead>
<tr>
<th>Code (Hex)</th>
<th>Returned Diagnostic Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Success</td>
</tr>
<tr>
<td>01</td>
<td>NM10 Microprocessor Memory Checksum Error</td>
</tr>
<tr>
<td>02</td>
<td>NM10 DMA Error</td>
</tr>
<tr>
<td>03</td>
<td>Transmitter Error</td>
</tr>
<tr>
<td>04</td>
<td>Receiver Error</td>
</tr>
<tr>
<td>05</td>
<td>Loopback Failure</td>
</tr>
</tbody>
</table>
## APPENDIX D

**TRANSMIT DATA FORMAT**

<table>
<thead>
<tr>
<th>BAR+ Φ</th>
<th>Destination Address A. (Byte 1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ 1</td>
<td>Destination Address B. (Byte 2)</td>
</tr>
<tr>
<td>+ 2</td>
<td>Dest. Addr. C. (Byte 3)</td>
</tr>
<tr>
<td>+ 3</td>
<td>Dest. Addr. D. (Byte 4)</td>
</tr>
<tr>
<td>+ 4</td>
<td>Dest. Addr. E. (Byte 5)</td>
</tr>
<tr>
<td>+ 5</td>
<td>Dest. Addr. F. (Byte 6)</td>
</tr>
<tr>
<td>+ 6</td>
<td>Type Field &lt;7:0&gt;</td>
</tr>
<tr>
<td>+ 7</td>
<td>Type Field &lt;15:8&gt;</td>
</tr>
<tr>
<td>+ 8</td>
<td>Data-First Byte</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>BAR+BCR-1</td>
<td>Data-Last Byte</td>
</tr>
</tbody>
</table>

41
APPENDIX E

RECEIVE DATA FORMAT

BAR + 0

<table>
<thead>
<tr>
<th>7</th>
<th>Frame Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ 1</td>
<td>Always 0</td>
</tr>
<tr>
<td>+ 2</td>
<td>Frame Length &lt;7:0&gt;</td>
</tr>
<tr>
<td>+ 3</td>
<td>Frame Length &lt;15:8&gt;</td>
</tr>
<tr>
<td>+4-9</td>
<td>Destination Address (6 Bytes)</td>
</tr>
<tr>
<td>+10-15</td>
<td>Source Address (6 Bytes)</td>
</tr>
<tr>
<td>+16</td>
<td>Type Field &lt;7:0&gt;</td>
</tr>
<tr>
<td>+17</td>
<td>Type Field &lt;15:8&gt;</td>
</tr>
<tr>
<td>+18</td>
<td>Data-First Byte</td>
</tr>
</tbody>
</table>

BAR + PRLTH + 3

<table>
<thead>
<tr>
<th>0</th>
<th>Data-Last Byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>CRC &lt;24:31&gt;</td>
<td></td>
</tr>
<tr>
<td>CRC &lt;16:23&gt;</td>
<td></td>
</tr>
<tr>
<td>CRC &lt;8:15&gt;</td>
<td></td>
</tr>
<tr>
<td>CRC &lt;0:7&gt;</td>
<td></td>
</tr>
</tbody>
</table>

BAR + BCR - 1

Note: Frame length is counted from first destination address byte up to and including the last CRC byte consecutively.
APPENDIX F

SOURCE CODE OF PROGRAM ETHTESTA.ASM

;*************************************************************************
;ETHERNET LEVEL ONE TEST PROGRAM--VERSION 1.13
;
;PROGRAM FILE NAME: ETHTESTA.COM– INVOKE COMMAND: ETHTESTA
;
;PROGRAM FUNCTION:(RUN CN 8080 BASED MDS SYSTEM)
;COMMANDS THE N13010 BOARD TO GO ONLINE, PERFORM ITS’
;DIAGNOSTIC TESTS THEN TRANSFERS A 42 BYTE DATA BLOCK FROM
;ADDRESS 0608 HEX TO ADDRESS 0812 HEX VIA THE MODULE INTER-
;FACE LOOPBACK MODE. TRANSFERRED DATA IS THEN DISPLAYED ON
;THE CONSOLE. THESE TESTS ONLY REQUIRE THE N13010 BOARD.
;THE CABLE TO THE TRANSCEIVER NEED NOT BE CONNECTED.
;
;TESTS PERFORMED:
; 1.) ONBOARD DIAGNOSTIC SELF TEST
; 2.) MODULE INTERFACE LOOPBACK TEST–VERIFIES THE
; FUNCTION OF THE N13010 LESS THE RECEIVE
; BUFFER.
;
;N13010 ETHERNET BOARD CONFIGURATION:
; 1.) JUMPER SET TO INTERRUPT LEVEL 5
; 2.) BASE PORT ADDRESS SWITCHES SET TO
; 1011 (0080H).
; 3.) PARALLEL PRIORITY TO AN ODD NUMBERED
; MULTIBUS SLCT.
;
;ORIGINAL PROGRAM: 03/10/83
;LAST REVISION: 04/30/83
;
;WRITER: MARK D. STOTZER
;
;ADVISOR: PROF. U.R. KODRES
;
;*************************************************************************
;MAIN PROGRAM:
; CRG EQU 100H
; N13010 REGISTER PORT ADDRESSES:
; CRG EQU 0080H; CMD REG LOCATION
; SREG EQU 00B1H; CMD STATUS REG LOCATION
; ISREG EQU 00B5H; INTERRUPT STATUS REG
IEREG EQU 0C38H: INTERRUPT ENABLE REG
BAR EQU 0C39H: EXTENDED ADDR REG
HBAR EQU 0CB8H: HIGH ADDR REG
LBAR EQU 0CBBH: LOW ADDR REG
LREG EQU 0CBEH: HIGHEST ADDR REG
LREG EQU 0C9EH: LOW BYTE COUNT REG

; OTHER NEEDED ADDRESSES:
BDOS EQU 0C05H: BDOS ENTRY POINT
CEREG EQU 07C9H: COPY OF INTERRUPT ENABLE REG
LSTM EQU 07C0H: ADDR OF INIT STACK PTR

; NECESSARY BDOS COMMANDS:
CONSIN EQU 01H: CONSOLE CHAR INPUT
CONSOUT EQU 02H: CONSOLE CHAR OUTPUT
PSTRING EQU 09H: PRINT TEXT STRING

; CLEAR COMMAND STATUS REGISTER BY READING IN SREG

; LOAD JUMP INSTRUCTION FOR INTERRUPT HANDLER: (INT 5)
MVI A, 0C3BH: JMP INST CODE
STA 0028H: LOAD IT IN ADDR 0028 HEX
LXI H, INTHDL
SHLD 029H

; OUTPUT INITIAL MESSAGE:
LXI D, BMSG
MVI A, 0C38H: ENABLE INTERRUPT 5-ETHERNET BOARD
OUT 0FDH
MVI A, 0DFFH
OUT 0FCH

; LOAD TRANSMIT DATA BLOCK—FIRST 3 BYTES ASSIGNED BY XEROX:
MVI A, 02H
STA 0000H
MVI A, 07H
STA 001FH
MVI A, 01H
STA 0025H

; LOAD INTERLAN ASSIGNED LAST 3 BYTES HERE:
DESTINP CALL CRLF
LXI D, DMSG0
MVI A, 03H: PRINT COMMAND
MVI A, 02H
CALL BDOS
CALL CRLF
LXI D, DMSG1
MVI C,PSTRING
CALL BDOS
CALL CRLF
LXI D,DMG2
MVI C,PSTRING
CALL BDOS
CALL CRLF
MVI C,CONSR:READY FOR CHOICE
CALL BDOS
CPI 31H
JZ DADDR2
CPI 32H
JZ DADDR1
CALL CRLF
LXI D,DMG3
MVI C,PSTRING
CALL BDOS
CALL CRLF
JMP DESTINP
DADDR1 CALL CRLF ;IF ADDR 00-03-EA SELECTED LOAD IT:
MVI A,00H
STA 0603H
MVI A,03H
STA 0604H
MVI A,0EAEH
STA 0605H
JMP ADDIN
DADDR2 CALL CRLF ;IF ADDR 00-04-0A SELECTED LOAD IT:
MVI A,00H
STA 0603H
MVI A,04H
STA 0604H
MVI A,0AH
STA 0605H
LOAD TYPE FIELD - 2 BYTES:
ADDIN MVI A,00H
STA 0606H
MVI A,00H
STA 0607H
;NOTE: FOR THIS TEST THE ACTUAL DATA IS IN ADDRESSES
;0608-0632HEX FOR TRANSMISSION
;*****************************************************
;READ IN THE TEST DATA:
MVI C,PSTRING
LXI D,DMG2
CALL BDOS
CALL CRLF
CALL CONIN
CALL CRLF
;GO ONLINE UPON POWER UP:
LXI SP,LASTM
EI
MVI A,09H;CMD TO GC ONLINE
OUT CREG
LXI D,OLMSG
MVI C,PSTRING
CALL BDOS
CALL CRLF
CALL READ

;RUN ONBOARD DIAGNOSTICS TEST:
MVI A,09H; CODE FOR SELF TEST COMMAND
OUT CREG
LXI D,STMSG
MVI C,PSTRING
CALL BDOS
CALL CRLF
CALL READ

;RUN MODULE INTERFACE LOOPBACK TEST:
MVI A,09H; GO BACK ONLINE
OUT CREG
LXI D,OLMSG
MVI C,PSTRING
CALL BDOS
CALL CRLF
CALL READ

;LOAD INTERRUPT ENABLE REGISTER=4. SET TO RECEIVE DATA.
DI
LXI H,CEREG
MVI A,04H
MOV M,A
OUT IEREG
EI

;RUN COMPLETE MODULE LOOP TEST:
MVI A,01H; ENTER MODULE LOOP TEST MODE
OUT CREG
LXI D,MLMSG
MVI C,PSTRING
CALL BDOS
CALL CRLF
CALL READ
CALL TRMSG;TRANSMIT TEST DATA BLOCK
LXI D,TRCMMSG
MVI C,PSTRING
CALL BDOS
CALL CRLF
CALL READ

;*************** TEST ONLY-MODULE LOOPBACK ***************
; THIS PATCH ENABLES DATA TRANSFER TO HOST MEMORY IN TEST
DI
MVI A,07H
LXI H,CEREG
MOV M,A

46
OUT IEREG
EI

;--------------------------------------------------------------------------------
MVI A, 03H; CLEAR LOOP TEST MODE
OUT CREG
LXI D, CLMSG
MVI C, PSTRING
CALL BDOS
CALL CRLF
CALL READ

;GO BACK ON-LINE
MVI A, 09H
OUT CREG
LXI D, OLMSG
MVI C, PSTRING
CALL BDOS
CALL CRLF
CALL READ

; DISPLAY DATA TRANSFERRED VIA ETHERNET BOARD TO CRT:
MVI C, PSTRING
LXI D, LMSG
CALL BDOS
CALL CRLF
CALL CONOUT
JMP 0; RETURN TO OPERATING SYSTEM

; END OF MAIN PROGRAM

;--------------------------------------------------------------------------------
; TRANSMIT SUBROUTINE:
TRMSG DI

; LOOP UNTIL INTERRUPT ENABLE REGISTER = 0 OR 4:
LOOP LXI H, CREG; CHECK IF N13010 BUSY
MOV A, M
CPI 00H
JZ CONT
CPI 04H
JZ CONT
EI
JMP LOOP
CONT DI; DISABLE INTS. AND CHECK AGAIN
LXI H, CREG
MOV A, M
CPI 00H
JZ CONT1
CPI 04H
JZ CONT1
EI
JMP LOOP
CONT1 MVI A, 00F
LXI H, CREG; DISABLE THE N13010 INTERRUPTS
MOV M, A
IEREG; SET INTERRUPT ENABLE REG = 2
EI

ADDR1 EQU 00H; LOCATION OF TRANSMIT DATA START=
ADDR2 EQU 06H; 600 HEX
ADDR3 EQU 03H

MVI A, ADDR1; LOAD TRANSMIT MESSAGE 1ST ADDR
OUT EBAR
MVI A, ADDR2
OUT HBAR
MVI A, ADDR3
OUT LBAR
MVI A, 06H; LOAD BYTE COUNT
OUT H3REG
MVI A, 032H
OUT L3REG
DI
MVI A, 06H; ENABLE Ni3010 TDD INTERRUPT
LXI H, C3REG
MOV M, A
OUT I3REG
EI

DONE MOV A, M; READ THE COPY OF IEREG-CEREG
CPI 06H
JZ DONE

TEST3 MVI A, 029E; LOAD TRANSMIT AND SEND COMMAND
OUT CREG
RET

; END TRANSMIT SUBROUTINE

; READ STATUS SUBROUTINE:
READ MVI B, 11111111B
MVI C, 00H
RDLP IN ISREG
ORA B
CPI 00FH
JNZ RDLP; CONTINUE LOOP UNTIL STATUS REG READ
IN SREG
CMP C
JNZ EFMSG
LXI D, MSG
MVI C, 09H
CALL EDOS
CALL CRLE
JMP RDONE

EFMSG LXI D, NMSG
MVI C, 09H
JMP EDOS
CALL CRLE

RDONE RET

; END READ SUBROUTINE:
;INTERRUPT HANDLER:
;SAVE CPU STATE:
INTHDL
EI
PUSH PSW
PUSH B
PUSH D
PUSH H
DI
LXI H,CEREG
MOV B,M; SAVE ENABLE REGISTER COPY VALUE
MVI A,06H
LXI H,CEREG; DISABLE NI3010 INTS.
OUT IREG
MOV M,A
MOV A,B
MVI B,04H; IS RBA INTERRUPT ENABLED?
CMP B
JZ RBA
MVI B,07H; IS RDD INTERRUPT ENABLED?
CMP B
JZ RDD
MVI A,04H; IF NEITHER OF ABOVE THEN WAS TDD
LXI H,CEREG; ENABLE RBA INTERRUPT
MOV M,A
OUT IREG
JMP FINI
RADD1 ECU 00H; 1ST ADDR TO WRITE RECVD FRAME TO=
RADD2 ECU 02H; 0800 HEX
RADD3 ECU 02H
RBA MVI A,RADD1; LOAD THE ADDRESS REGISTERS
OUT EBA
MVI A,RADD2
OUT E3AR
MVI A,RADD3
OUT LBAR
MVI A,06H; NOW LOAD BYTE COUNT REGISTERS
OUT HREG
MVI A,040H
OUT LBREG
LXI H,CEREG
MVI A,07H; ENABLE RDD INTERRUPT
MOV M,A
OUT IREG
JMP FINI
RED LXI H,CEREG
MVI A,04H
;RECEIVE PROCESS W/KE UP IN HERE
MOV M,A
OUT IREG
FINI
;RESTORE CPU STATE:
POP E
POP D
POP B
DI
MVI A, 020H; RESTORE INTERRUPT STATUS
OUT 0FDH
POP PSW
EI
RET

;END INTERRUPT HANDLE?

;**************************************************************************
CRLF MVI C, CONSOUT; GENERATES CARRIAGE RETURN -LINE
MVI E, 0DH
CALL BDOS
MVI C, CONSOUT
MVI E, 0AH
CALL BDOS
RET

;**************************************************************************
CONIN LXI H, 0609H; READ TEST DATA INPUT FROM CONSOLE
INLP MVI C, CONIN
PUSH E
CALL BDOS
POP H
MOV M, A
CPI 60H; COMPARE TO GRAVE ACCENT
RZ
INX H
JMP INLP

;**************************************************************************
CONOUT LXI H, 08127; OUTPUT TEST DATA TO THE CONSOLE
OTLP MVI C, CONSOUT
MOV E, M
MOV A, E
CPI 60H; IF GRAVE ACCENT THEN RETURN
RZ
PUSH H
CALL BDOS
POP H
INX E
JMP OTLP

;**************************************************************************
BMSG DB 'ETHERNET LEVEL ONE TEST PROGRAM: VERSION:
DB 'ION: 1.13: 04/30/83-MDS$'
OLMSG DB 'ONLINE COMMAND ISSUED'
STMSG DB 'SELF TEST COMMAND ISSUED$
MLMSG DB 'MODULE LOOPBACK COMMAND ISSUED$
CLMSG DB 'CLEAR LOOPBACK COMMAND ISSUED$
TRCMSG DB 'TRANSMIT/SEND COMMAND ISSUED$
MSG DB 'COMMAND EXECUTED$
NMSG DB 'COMMAND FAILED$'
FMSG DB 'ENTER TEXT (42 CHAR MAX) FOR MODULE'
DB 'INTERFACE LOOPBACK (42 CHAR MAX)
DB (END WITH A GRAVE ACCENT => )$'
LMSG DB 'THE DATA TRANSFERRED VIA MODULE INTERFACE'
DB 'FACE LOOPBACK IS: $'
DMSG1 DB 'ENTER ADDRESS OF INSTALLED NI3010'
DB 'BOARDS$'
DMSG2 DB 'BOARD 00-04-0A: ENTER 1 $'
DMSG3 DB 'BOARD 00-03-FA: ENTER 2 $'
DMSG3 DB 'INCORRECT SELECTION-TRY AGAIN: $'

;**********************************************************
;**********************************************************
END; ETHERNET LEVEL ONE TEST PROGRAM-VERSION 1.13
APPENDIX G

SOURCE CODE OF PROGRAM ETHTEST3.ASM

`;**********************************************************************
`;ETHERNET SECOND LEVEL TEST PROGRAM--VERSION 2.04
`;**********************************************************************

PROGRAM FILE NAME: ETHTESTB.COM- INVOKE COMMAND: ETHTESTB

;PROGRAM FUNCTION:(RUN ON 8000 BASED MDS SYSTEM)
;SELF TEST.IT THEN TRANSFERS A 42-BYTE BLOCK OF TEXT FROM A
;BLOCK OF MEMORY STARTING AT ADDRESS 0700 HEX TO ANOTHER
;BLOCK AT 0900 HEX IN TWO SEPARATE TESTS VIA THE NI3010
;BOARD. SUCCESSFUL COMPLETION OF THESE TESTS VERIFIES THE
;FUNCTIONING OF ALL THE HARDWARE NECESSARY TO COMMUNICATE
;WITH OTHER HOSTS ON THE NETWORK.

;TESTS PERFORMED:
; 1.) BOARD DIAGNOSTIC SELF TEST
; 2.) MODULE INTERFACE LOOPBACK-VERIFIES THE
;      FUNCTIONING OF THE NI3010 BOARD INCLUDING THE
;      NMIO PROTOCOL MODULE.
; 3.) EXTERNAL LOOPBACK-VERIFIES THE FUNCTIONING OF
;      ABOVE AND THE FLAT CABLE,TRANSCEIVER AND
;      NETWORK COAXIAL CABLE.

;NI3010 ETHERNET BOARD CONFIGURATION:
; 1.) JUMPER SET TO INTERRUPT LEVEL 5.
; 2.) BASE PORT ADDRESS SWITCHES SET TO
;      1211 (00000).
; 3.) PARALLEL PRIORITY TO AN ODD NUMBERED
;      MULTIBUS SLOT.

;ORIGINAL PROGRAM: 03/31/83
;LAST REVISION: 04/30/83
;WRITER: MARK D. STOTZER
;ADVISOR: PROF. U.R. KOTRES

;**********************************************************************

;MAIN PROGRAM:
; ORG 120H
; NI3010 REGISTER PORT ADDRESSES:

52
CREG EQU $0030H; CMD REG LOCATION
SREG EQU $001FH; CMD STATUS REG LOCATION
ISREG EQU $0025H; INTERRUPT STATUS REG
IEREG EQU $0029H; INTERRUPT ENABLE REG
EBAR EQU $0029H; EXTENDED BASE ADDR REG
HBAR EQU $0030H; HIGH BASE ADDR REG
LBAR EQU $002BH; LOW BASE ADDR REG
HSREG EQU $003CH; HIGH BYTE COUNT REG
LBREG EQU $003DH; LOW BYTE COUNT REG

; OTHER NEEDED ADDRESSES:
EDOS EQU $005F; BDOS ENTRY POINT
CEFG EQU $0006H; COPY OF INTERRUPT ENABLE REG
STATUS EQU $001AH; COPY OF CMD STATUS REG

; NEEDED BDOS COMMANDS:
PSTRING EQU $05; PRINT STRING FUNCTION

CONIN EQU $01H; CONSOLE CHAR INPUT FUNCTION
CONOUT EQU $02H; CONSOLE CHAR OUTPUT FUNCTION

; READ CMD STATUS REG ON POWER UP: REQUIRED FOR INITIALIZATION
IN SREG

; OUTPUT INITIAL MESSAGE TO USER:
LXI D, PMSG
MVI C, PSTRING
CALL EDOS
CALL CPLF

; LOAD JUMP INSTRUCTION FOR INTERRUPT HANDLER: (INT 5)
MVI A, $03FH; JMP INST CODE
STA $029F; LOAD IT IN ADDR $0298 HEX
LXI H, INTHDL
SHLD $029F

; SET UP INTERRUPT CONTROL: (INT 5)
MVI A, $012H
OUT $07DH
MVI A, $0DFH; ENABLE INTERRUPT 5-ETHERNET BOARD
OUT $0FH

; LOAD TRANSMIT DATA BLOCK-FIRST 3 BYTES ASSIGNED BY XEROX:
MVI A, $02H
STA $0700H
MVI A, $07H
STA $0701H
MVI A, $01F
STA $0702H

; LOAD INTERLAN ASSIGNED LAST 3 BYTES HERE:
DESTINP
CALL CPLF
LXI D, PMSG0; ASK USER TO INPUT THIS ADDRESS
MVI C, PSTRING

53
CALL BDOS
CALL CRLF
LXI D, DMSG1
MVI C, PSTRING
CALL BDOS
CALL CRLF
LXI D, DMSG2
MVI C, PSTRING
CALL BDOS
CALL CRLF
MVI C, CONSELIN; READ USER INPUT OF ADDRESS
CALL BDOS
CPI 31H
JZ DADDR2
CPI 32H
JZ DADDR1
CALL CRLF
LXI D, DMSG3
MVI C, PSTRING
CALL BDOS
CALL CRLF
JMP DESTINP
DADDR1
CALL CRLF; ADDR 00-03-EA SELECTED BY USER: LOAD
MVI A, 00H
STA 0703H
MVI A, 03H
STA 0704H
MVI A, 0EH
STA 0705H
JMP ADDIN
DADDR2
CALL CRLF; ADDRESS 00-04-CA SELECTED: LOAD IT
MVI A, 00H
STA 0703H
MVI A, 04H
STA 0704H
MVI A, 04H
STA 0705H
LOAD TYPE FIELD - 2 BYTES:
ADDIN
MVI A, 00H
STA 0626H
MVI A, 04H
STA 0627H
; NOTE: FOR THIS TEST THE ACTUAL DATA IS IN ADDRESSES
; 0608-0632HEX FOR TRANSMISSION
;*****************************************************************************
; READ IN THE TEST DATA FOR MODULE INTERFACE LOOPBACK TEST:
MVI C, PSTRING
LXI D, FMSG
CALL BDOS
CALL CRLF
CALL CONIN
CALL CRLF

;GO ONLINE UPON POWER UP:
EI
MVI A,09H: CMD TO GO ONLINE
OUT CFEG
CALL READ

;***************************************************************************

;RUN ONBOARD DIAGNOSTICS TEST:
MVI A,0AH; CODE FOR SELF TEST COMMAND
OUT CREG
CALL READ

;***************************************************************************

;LOAD INTERRUPT ENABLE REGISTER=4. SET TO RECEIVE DATA.
DI
LXI M,CEREG
MVI A,04H
MOV M,A
OUT IEREG
EI

;COMMAND MODULE INTERFACE LOOPBACK MODE:
MVI A,02H
OUT CREG
CALL READ

;TRANSFER THE TEST DATA:
CALL TRMSG
CALL READ

;DISPLAY DATA TRANSFERRED BY MODULE INTERFACE LOOPBACK TEST:
MVI C,PSTRING
LXI D,LMSG
CALL BDOS
CALL CFLF
CALL COUT; TEXT OUTPUT TO THE CONSOLE
CALL CRLF

;***************************************************************************

;PERFORM INTERNAL LOOPBACK TEST:
;READ IN TEST DATA FOR EXTERNAL LOOPBACK TEST:
MVI C,PSTRING
LXI D,FEMSG
CALL BDOS
CALL CFLF
CALL COUT
CALL CRLF

;EXIT INTERNAL LOOP TEST MODE:
MVI A,03H
OUT CFEG
CALL READ

;GO BACK ONLINE:
;TRANSMIT THE TEST DATA:
MVI C, PSTRING
LXI D, LEMSG
CALL BDOS
CALL CPLF
CALL CONOUT
CALL CRLF
JMP $ ;RETURN TO OPERATING SYSTEM

; DISPLAY DATA TRANSFERRED VIA INTERNAL LOOPBACK TO CRT:
MVI CPSTRING
LXI D, LEMSG
CALL BDOS
CALL CPLF
CALL CONOUT
CALL CRLF

; END OF MAIN PROGRAM

;Transmit Subroutine:
TRMSG
DI
;LOOP UNTIL INTERRUPT ENABLE REGISTER = 0 CR 4:
LOOP
LXI H, CEREG
MOV A, M
CPI 00H
JZ CONT
CPI 04H
JZ CONT
EI
JMP LOOP
CONT
LXI H, CEREG
MOV A, M
CPI 0CH
JZ CONT1
CPI 04H
JZ CONT1
EI
JMP LOOP
CONT1
MVI A, 0CH
LXI H, CEREG
MOV M, A
OUT IEREG; SET INTERRUPT ENABLE REG = 0
EI
ADDR1 EQU 00H; LOCATION OF TRANSMIT BUFFER TOP
ADDR2 EQU 07H
ADDR3 EQU 00H
MVI A, ADDR1; LOAD TRANSMIT MESSAGE 1ST ADDR.
OUT EBAR
MVI A, ADDR2
OUT EBAR
MVI A, ADDR3
OUT EBAR

56
MVI A,00H; LOAD BYTE COUNT
OUT HERE
MVI A,032H
OUT LBREG
DI
MVI A,06H; ENABLE TDD INTERRUPT
LXI H,CEREG
MOV M,A
OUT IEREG
EI
DONE
LXI H,CEREG
MOV A,0FH; READ THE COPY OF IEREG=CEREG
CPI 0FH
JZ DONE

TEST3
MVI A,029H; LOAD TRANSMIT AND SEND COMMAND
OUT CREg
RET

; END TRANSMIT SUBROUTINE
;******************************************************************************

; READ STATUS SUBROUTINE:
READ
MVI B,11111110B
MVI C,00H
RDLP
IN ISREG
ORA B
CPI 0FFH
JNZ RDLP; CONTINUE LOOP UNTIL STAT REG READY
IN SFEG
LXI H,STATUS; KEEP COPY OF CMD STAT REG
MOV M,A
CMP C
JNZ !PMSC
LXI D,MSG
MVI C,?STRING
CALL 'BDOS
CALL CRLF
JMP 'RDONF
LXI D,NMSG
MVI C,PSTRING
CALL EDOS
MVI B,250H
LXI H,STATUS
MOV A,M
ADD B
MVI C,?CONSOl;ERROR CODE TO CONSOLE
MOV F.A
CALL 'BDOS
CALL CRLF
LXI D,NMSG1
MVI C,PSTRING
CALL 'BDOS
CALL CRLF
RDONE PET
;
; END READ SUBROUTINE:
;***************************************************************************
; INTERRUPT HANDLER:
;SAVE CPU STATE:
INTHDL
DI
PUSH PSW
PUSH B
PUSH D
PUSH H
EI
LXI H,CLREG
MOV B,M; SAVE ENABLE REGISTER COPY VALUE
MVI A,00H; DISABLE NI3010 INTERRUPTS
LXI H,CLREG
MOV M,A
OUT IEREG
MOV A,B
MVI B,04H; WAS RBA INTERRUPT ENABLED?
CMP B
JZ RBA
LXI H,CLREG; WAS TDD-NCV ENABLE RBA AGAIN
MOY M,A
OUT IEREG
JMP FINI

RADD1 EQU 02H; LOCATION OF WHERE TO WRITE RECVD
RADD2 EQU 09H; FRAME DATA IN HOST MEMORY
RADD3 EQU 0EH
RBA
MVI A,RADD1; NOW LOAD ADDR INTO ADDR REGS.
OUT EBAR
MVI A,RADD2
OUT EBAR
MVI A,RADD3
OUT LBAR?
MVI A,00H; LOAD BYTF COUNT REGISTERS
OUT HBRFG
MVI A,040H
OUT LPRYG
LXI H,CLREG
MVI A,07H; ENABLE RDD INTERRUPT
MOY M,A
OUT IEREG
JMP FINI

RDD
LXI H,CFREG
MVI A,04H

; RECEIVE PROCESS WAKE UP IN HERE
MOV M,A

58
OUT IREG
FINI  EI

; RESTORE CPU STATE:
POP H
POP D
POP B
DI
MVI A, 22H; RESTORE INTERRUPT STATUS
OUT 0FDH
POP PSW
EI
RET

; END INTERRUPT HANDLER

CELF MVI C, CONSCOUT; GENERATES CARRIAGE RTN + LFEED
MVI F, 0DH
CALL BDOS
MVI C, CONSCUT
MVI F, 0AH
CALL BDOS
RET

; *********************************************************
CONIN LXI H, 0708H; READ TEST DATA INPUT FROM CONS.
INLP MVI C, CONSCIN
PUSH H
CALL BDOS
POP H
MOV M, A
CPI 60H; IF GRAVE ACCENT THEN RETURN
RZ
INX H
JMP INLP

; *********************************************************
CONOUT LXI H, 0912H; OUTPUT TEST DATA TO THE CONSOLE
OTLP MVI C, CONSCOUT
MOV Y, M
MOV A, E
CPI 60H; TEST FOR END CHAR-GRAVE ACCENT
RZ
PUSH H
CALL BDOS
POP H
INX H
JMP OTLP

; *********************************************************
BMSG DB 'ETHERNET SECOND LEVEL TEST PROGRAM:
  VEPStION 2.04: 24/30/83-MDS'
DMSG0 DB 'ENTFR ADDRESS OF INSTALLED NI 3010 '
  DB '30PD: ' $'
DMSG1 DB 'CARD #4-C4: ENTER ' 1 ' $'
DMSG2 DB 'CARD 00-03-CA: ENTER ' 2 ' $'

59
<table>
<thead>
<tr>
<th>LETTER</th>
<th>NATURE OF FAILURE</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>YOU ISSUED AN INAPPROPRIATE COMMAND THE BOARD IS IN.</td>
</tr>
<tr>
<td>T</td>
<td>BOARD TIMER TIMED OUT—POSSIBLE PROBLEM.</td>
</tr>
<tr>
<td>U</td>
<td>TRANSMIT BUFFER SIZE EXCEEDED:</td>
</tr>
<tr>
<td>V</td>
<td>FRAME SENT TO BOARD TOO SMALL:</td>
</tr>
<tr>
<td>X</td>
<td>EXCESSIVE COLLISIONS</td>
</tr>
</tbody>
</table>

END;ETHERNET SECOND LEVEL TEST PROGRAM—VERSION 2.
APPENDIX H

SOURCE CODE OF MAIN MODULE ETHERNET.PLI

ETHERNET:/*MAIN MODULE—APPLICATION LAYER—ISO LEVEL 7*/

PROCEDURE OPTIONS (MAIN);

DECLARE

/* LOCAL VARIABLES */
COUNT7 FIXED BINARY(7), /*LOOP CONTROL VARIABLE*/
COUNT7A FIXED BINARY(7), /*LOOP CONTROL*/
COUNT7B FIXED BINARY(7), /*LOOP CONTROL*/
COUNT7C FIXED BINARY(7), /*LOOP CONTROL*/
DSKNO CHARACTER(1), /*USER INPUT DISK NUMBER*/
FRAMD CHARACTER(1), /*USER INPUT FRAME SIZE*/
SELECT CHARACTER(1), /*USER INPUT MODE SELECTION*/

/* GLOBAL VARIABLES */
RECFILE FIXED BINARY(7) EXTERNAL, /*RECV FILE NO.*/
FSIZE FIXED BINARY(15) EXTERNAL, /*FRAME SIZE*/
TERM FIXED BINARY(7) EXTERNAL, /*TERMINAL FLAG*/
TRMSEL FIXED BINARY(7) EXTERNAL, /*CMD MODE FLAG*/

/* GLOBAL DATA STRUCTURES */
TXBUF(1508) FIXED BINARY(7) EXTERNAL, /*TRANS BUFF*/
RXBUF(1522) FIXED BINARY(7) EXTERNAL, /*RECV BUFF*/
TXTBUF (128) FIXED BINARY(7) EXTERNAL, /*TEXT BUFF*/

1 RXFCB EXTERNAL, /*RECEIVE FILE CONTROL BLOCK*/
2 DISK FIXED BINARY(7),
2 FTYP CHARACTER(3),
2 RFCB(24) FIXED BINARY(7).

1 TXFCB EXTERNAL, /*TRANSMIT FILE CONTROL BLOCK*/
2 DISK FIXED BINARY(7),
2 FTYP CHARACTER(3),
2 TXFCB(24) FIXED BINARY(7).

/* EXTERNAL MODULES */
INIT ENTRY; /* INITIALIZES INTERRUPTS & NI3010*/
SENDATA ENTRY; /* TRANSMIT ISO LEVEL 6 MODULE */
RECEIVE ENTRY; /* RECEIVE MODULE */

/*LAST REVISION: 09/15/83—0900 ORIGINAL PROGRAM:07/29/83 */
/*AUTHOR: CAPT. MARK D. STOTZER-USMC—AESIS GROUP */
/*THESIS ADVISOR: PROFESSOR UNO R. ECRIS—COMP. SCIENCE */

PUT SKIP LIST('URRENT VERSIO-UTHERETN COMMUNICATION PROGRAM—VERSIO N 5.0');

PUT SKIP LIST('ALLOWS THIS HOST TO CONNECT TO THE NET.');
PUT SKIP LIST('CNTL-H=BACKSPACE FOR TEXT ENTRIES:');
PUT SKIP LIST('***************************************************************************');
PUT SKIP(2);
RECFIL=47;
COUNT7=1;
DO WHILE (COUNT7=1);
  COUNT7A=1;
  DO WHILE(COUNT7A=1);
    PUT SKIP(2);
    PUT SKIP LIST('*************** MAIN MENU ***************');
    PUT SKIP LIST('WRITE RECEIVED FILES TO DISK NO:');
    PUT SKIP LIST('DEFAULT DRIVE(A) = 1');
    PUT SKIP LIST('DISK DRIVE A = 2');
    PUT SKIP LIST('DISK DRIVE B = 3');
    PUT SKIP LIST('ENTER DRIVE NUMBER=');
    GET LIST(DSKNO);
    PUT SKIP(2);
    IF DSKNO='1' THEN DO;
      RXFCB.DISK=0; /* LOAD DISK NUMBER IN FCB */
      COUNT7A=2;
    END;
    ELSE IF DSKNO='2' THEN DO;
      RXFCB.DISK=1; /* DISK NUMBER TO FCB */
      COUNT7A=2;
    END;
    ELSE IF DSKNO='3' THEN DO;
      RXFCB.DISK=2; /* DISK NUMBER TO FCB */
      COUNT7A=2;
    END;
    ELSE PUT SKIP LIST('INVALID DRIVE NUMBER-REENTER:');
  END; /*DO LOOP*/
  COUNT7B=1;
  DO WHILE (COUNT7B=1);
    PUT SKIP LIST('ETHERNET FRAME DATA BLOCK SIZE:');
    PUT SKIP LIST('SELECT 128 FOR ALL FILE OPERATIONS');
    PUT SKIP LIST('AND VAX COMMUNICATIONS.');
    PUT SKIP LIST(' 128 BYTES = 1');
    PUT SKIP LIST(' 256 BYTES = 2');
    PUT SKIP LIST(' 512 BYTES = 3');
    PUT SKIP LIST(' 1024 BYTES = 4');
    PUT SKIP LIST(' 1500 BYTES = 5');
    PUT SKIP LIST('ENTER SELECTION=>');
26
GET LIST (FRAMD);  
PUT SKIP(2);  
IF FRAMD = '1' THEN  
    DO:  
      FRSIZE = 128; /* SET THE FRAME SIZE */  
      COUNT7B = 2;  
    END;  
ELSE  
    IF FRAMD = '2' THEN  
        DO:  
          FRSIZE = 256; /* SET FRAME SIZE */  
          COUNT7B = 2;  
        END;  
    ELSE  
      IF FRAMD = '3' THEN  
        DO:  
          FRSIZE = 512; /* SET FRAME SIZE */  
          COUNT7B = 2;  
        END;  
      ELSE  
        IF FRAMD = '4' THEN  
          DO:  
            FRSIZE = 1024; /* SET THE FRAME SIZE */  
            COUNT7B = 2;  
          END;  
        ELSE  
          IF FRAMD = '5' THEN  
            DO:  
              FRSIZE = 1536; /* SET FRAME SIZE */  
              COUNT7B = 2;  
            END;  
          ELSE  
            PUT SKIP LIST ("INCORRECT CHOICE-REENTER:");  
          END;  
      END;  
    END;  
ELSE  
  PUT SKIP LIST ("INCORRECT CHOICE-REENTER:");  
END; /* DO LOOP */  
VTERM = 0; /* RESET TERMINAL FLAG TO FALSE */  
TPMODE = 0; /* RESET COMMAND MODE FLAG TO FALSE */  
CALL INIT;  
PUT SKIP LIST ("OPERATING MODES:");  
PUT SKIP LIST ("*****************************************");  
PUT SKIP LIST ("RECEIVE WAIT LOOP = 1");  
PUT SKIP LIST ("TRANSMIT FILE OR MESSAGE = 2");  
PUT SKIP LIST ("VIRTUAL TERMINAL OF VAX = 3");  
PUT SKIP LIST ("VAX COMMAND MODE = 4");  
PUT SKIP LIST ("DISCONNECT FROM NET = 5");  
PUT SKIP LIST ("*****************************************");  
PUT SKIP LIST ("ENTER SELECTION =>");  
GET LIST (SELECT);  
PUT SKIP (2);  
IF SELECT = '1' THEN /* RECEIVE MODE */  
      DO:  
        TXBUFF(1) = 2; /* LOAD FIRST THREE DEST ADDR BYTES */  
      END;
TXBUFF(2)=7;/ * FOR ACK REPLY IN RECEIVE MODE */
TXBUFF(3)=1;
PUT SKIP LIST('IN RECEIVE WAIT LOOP-TO RETURN TO' );
PUT SKIP LIST('MAIN MENU: ENTER <CR> ==>' );
PUT SKIP LIST('*******************************' );
PUT SKIP (2);
CALL RECEIVE;
END;
ELSE
IF SELECT='2' THEN /* NORMAL TRANSMIT */
CALL TRANS2 ;
ELSE
IF SELECT='3' THEN /* VAX TERMINAL MODE */
DO;
VT ERM=1;/*** SET THE TERMINAL FLAG TO TRUE */
FRSIZE=1500;
PUT SKIP LIST('****** VAX TERMINAL MODE ********' );
PUT SKIP (1);
PUT SKIP LIST('VAX TERMINAL SERVICE:' );
PUT SKIP LIST('DATA BLOCK SIZE PER FRAME=' );
PUT LIST (FRSIZE);
PUT SKIP LIST('--------------------------------------' );
PUT SKIP LIST('TERMINAL ENTRY BY LINE OF TEXT' );
PUT SKIP LIST('BEGIN AFTER INITIAL V PROMPT: "V"' );
PUT SKIP LIST('ENTER: TEXT LINE<CR>' );
PUT SKIP LIST('PROMPT WILL AUTOMATICALLY REAPPEAR' );
PUT SKIP LIST('OF THE NEXT LINE YOU BEGIN.' );
PUT SKIP LIST('--------------------------------------' );
PUT SKIP LIST('TO END TERMINAL SESSION:' );
PUT SKIP LIST('ENTER: "<CR> AFTER "V" "' );
PUT SKIP LIST('--------------------------------------' );
PUT SKIP (1);
TXBUFF(1)=2; /* LOAD THE VAX NET ADDR INTO THE SIX*/
TXBUFF(2)=7; /* ADDRESS BYTES */
TXBUFF(3)=1;
TXBUFF(4)=0;
TXBUFF(5)=7;
TXBUFF(6)=127;
TXBUFF(7)=0;/* LOAD THE TYPE TWO TYPE FIELD BYTES */
TXBUFF(8)=0;
COUNT7C=1;
PUT SKIP LIST('V>' );
DO WHILE (COUNT7C=1);
CALL SENDATA;
PUT SKIP LIST('V>' );
IF VTERM=0 THEN /* END TERMINAL SESSION*/
DC:
PUT SKIP LIST('**** END TERMINAL SESSION ****' );
COUNT7C=2;
END;
ELSE
    DO;
        CALL INIT;
        CALL RECEIVE;
        PUT LIST("**H"H"HV**");
    END;
END; /* DO LOOP */
END;
ELSE
    IF SELECT='4' THEN /* VAX COMMAND MODE */
        DO;
            PUT SKIP LIST("*** VAX COMMAND INSTRUCTIONS ***");
            PUT SKIP LIST("TO DOWNLOAD A FILE FROM THE VAX: ");
            PUT SKIP LIST("ENTER THE MESSAGE:");
            PUT SKIP LIST("" IFNAME(VAX).FTYPE(VAX)/XXX" "");
            PUT SKIP LIST("WHERE "XXX"= EXE FOR NON-TEXT FILES");
            PUT SKIP LIST("AND "XXX"=TXT FOR TEXT FILES");
            PUT SKIP LIST("FILE WILL THEN BE IMMEDIATELY SENT");
            PUT SKIP LIST("TO THIS HOST.");
            PUT SKIP LIST("----------------------------------");
            PUT SKIP LIST("TO UPLOAD A FILE TO THE VAX:");
            PUT SKIP LIST("1.) ENTER THE MESSAGE:");
            PUT SKIP LIST("** QFNAME(VAX).FTYPE(VAX)/XXX" ");
            PUT SKIP LIST("TO OPEN A VAX FILE BY THE ABOVE NAME");
            PUT SKIP LIST("2.) THEN:");
            PUT SKIP LIST("SEND THE FILE TO THE VAX ADDRESS USING");
            PUT SKIP LIST("THE NORMAL FILE SENDING SELECTIONS.");
            PUT SKIP LIST("----------------------------------");
            PUT SKIP LIST("TO UPLOAD A FILE TO THE VAX:");
            PUT SKIP LIST("1.) ENTER THE MESSAGE:");
            PUT SKIP LIST("** QFNAME(VAX).FTYPE(VAX)/XXX" ");
            PUT SKIP LIST("TO OPEN A VAX FILE BY THE ABOVE NAME");
            PUT SKIP LIST("2.) THEN:");
            PUT SKIP LIST("SEND THE FILE TO THE VAX ADDRESS USING");
            PUT SKIP LIST("THE NORMAL FILE SENDING SELECTIONS.");
            PUT SKIP LIST("----------------------------------");
        END;
        TRMODE=1; /*SET VAX CMD MODE FLAG TO TRUE*/
        FRSIZE=128;
        TXBUFF(1)=2; /*LOAD THE VAX NET ADDR INTO THE SIX */
        TXBUFF(2)=7; /*ADDRESS BYTES */
        TXBUFF(3)=1;
        TXBUFF(4)=0;
        TXBUFF(5)=7;
        TXBUFF(6)=127;
        TXBUFF(7)=0; /*LOAD THE TWO TYPE FIELD BYTES */
        TXBUFF(8)=0;
        CALL SENDATA;
        CALL INIT;
        RXBUFF(17)=255;
        CALL RECEIVE;
        END;
    ELSE
        IF SELECT='5' THEN /* DISCONNECT BY EXITING TO CP/M */
            COUNT7=2;
        ELSE
            PUT SKIP LIST("INCORRECT OPMODE SELECTION-REENTER:");
        END; /* DO LOOP */
PUT SKIP LIST(‘DISCONNECTING FROM NET-RETURNING TO C?/M.’);

TRANS2: /* GETS USER INPUT OF FILE DATA */

PROCEDURE:

DECLARE
    /* LOCAL VARIABLES */
    COUNT6 FIXED BINARY(7), /* LOOP CONTROL*/
    COUNT6A FIXED BINARY(7), /* LOOP CONTROL*/
    COUNT6B FIXED BINARY(7), /* LOOP CONTROL*/
    COUNT6C FIXED BINARY(7), /* LOOP CONTROL*/
    SENDTYPE CHARACTER(1), /* USER INPUT TRANSMIT TYPE*/
    FTTP CHARACTER(1), /* USER INPUT FILETYPE*/
    DNRO CHARACTER(1), /* USER INPUT DRIVE NO. */
    /* FILE DATA ENTRY DCLS */
    I FIXED,
    FN CHARACTER(20),
    LOWER CHARACTER(26) STATIC INITIAL
    (‘abcdefghijklmnopqrstuvwxyz’),
    UPPER CHARACTER(26) STATIC INITIAL
    (‘ABCDEFGHIJKLMNOPQRSTUVWXYZ’),
    /* GLOBAL VARIABLES */
    FILTYP FIXED BINARY (7) EXTERNAL, /* FILE NATURE*/
    FNOP FIXED BINARY (7) EXTERNAL, /*FILE NOT OPEN FLG*/
    /* GLOBAL DATA STRUCTURES */
    TXBUFF(150) FIXED BINARY(?) EXTERNAL, /* TRANSMIT BUFF*/
    1 TXFCB EXTERNAL, /* TRANSMIT FILE CONTROL BLOCK*/
    2 DISK FIXED BINART(7),
    2 FNAME CHARACTER(8),
    2 FTTP CHARACTER(3),
    2 TFCB(24) FIXED BINART(?)
    /* EXTERNAL MODULES */
    SENDATA ENTRY; /* ISO LEVEL 3 FRAME SENDER*/

COUNT6 =1;
DO WHILE(COUNT6=1);
    PUT SKIP LIST(‘TRANSMISSION OPTIONS:’);
    PUT SKIP LIST(‘SEND A MESSAGE = 1’);
    PUT SKIP LIST(‘SEND A DISK FILE = 2’);
    PUT SKIP LIST(‘ENTER SELECTION ==> ’);
    GET LIST(SENDTYPE);
    PUT SKIP (2);
    TXBUFF(6)=0; /* TYPE FIELD BYTE 2=NORMAL MSG OR FILE*/
    IF SENDTYPE=’1’ THEN /* SEND A MESSAGE */
    DO;
        TXBUFF(7)=2; /* TYPE FIELD BYTE 1=MESSAGE*/
        CALL SENDATA;
        COUNT6=2;
    END;
ELSE
IF SENDTYPE='2' THEN /*SEND A DISK FILE*/
DO;
  TXBUFF(?)=15;/* TYPE FIELD BYTE 1= FILE*/
  COUNT6A=1;
  DO WHILE(COUNT6A=1);
    PUT SKIP LIST('NATURE OF FILE TO SEND:');
    PUT SKIP LIST('TEXT (ASCII) FILE = 1');
    PUT SKIP LIST('MACHINE CODE (COM) FILE = 2');
    PUT SKIP LIST('**********');
    PUT SKIP LIST('ENTER TYPE OF FILE CHOICE ==>');
    GET LIST(FTTP);
    PUT SKIP(2);
    IF FTTP='1' THEN
      DO;
        FILTYP=1;/* SET THE FILETYP=TEXT FILE */
        COUNT6A=2;
      END;
      ELSE
        IF FTTP='2' THEN
          DO;
            FILTYP=2;/* FILE TYPE=MACHINE FILE */
            COUNT6A=2;
          END;
        ELSE
          PUT SKIP LIST('INCORRECT CHOICE-REENTER:');
          END;/* DO LOOP */
        COUNT6B=1;
        DO WHILE(COUNT6B=1);
          COUNT6C=1;
          DO VEIL(COUNT6C=1);
            PUT SKIP LIST('SPECIFY FILE TO SEND:');
            PUT SKIP LIST('FILE LOCATED ON:');
            PUT SKIP LIST('DRIVE A = 1');
            PUT SKIP LIST('DRIVE B = 2');
            PUT SKIP LIST('**********');
            PUT SKIP LIST('ENTER DRIVE NUMBER==>');
            GET LIST(DRNO);
            PUT SKIP(2);
            IF DRNO='1' THEN
              DO;
                TXFCB.DISK=1;
                COUNT6C=2;
              END;
              ELSE
                IF DRNO='2' THEN
                  DO;
                    TXFCB.DISK=2;
                    COUNT6C=2;
                  END;
                ELSE
                  ...
PUT SKIP LIST('INVALID DRIVE-REENTER:\');
END;/* DO LOOP */
PUT SKIP LIST('ENTER: "FILENAME.FILETYPE"=>');
GET LIST(FN);
PUT SKIP(2);
FN=TRANSLATE(FN,UPPER,LOWER);
I=INDEX(FN,'.');
IF I=0 THEN
  DO;
    TXFCB.FNAME=FN;
    TXFCB.FTYPE='.';
  END;
ELSE
  DO;
    TXFCB.FNAME=SUBSTR(FN,1,I-1);
    TXFCB.FTYPE=SUBSTR(FN,I+1);
  END;
  TXFCB.TFCB(1)=0;/* SET FCB FIELDS THAT COUNT=\*/
  TXFCB.TFCB(4)=0;/* CURRENT EXTENT, RECORD ETC. */
  TXFCB.TFCB(21)=0;
  CALL SENDAT4;
  IF FNOP=1 THEN
    COUNT6=2;
  END;/* DO LOOP */
  COUNT6=2;
END;
ELSE
  PUT SKIP LIST('INCORRECT TRANSMIT MODE-REENTER:\');
END;/* DO LOOP */
END TRANS2:

END ETHERNET;/* ISO LAYER 7 MODULE */
APPENDIX I
SOURCE CODE FOR MODULE SENDATA.PLI

SENDATA: /* PRESENTATION LAYER MODULE-ISO LEVEL 6 */

PROCEDURE;

DECLARE
/* LOCAL VARIABLES */
COUNT5A FIXED BINARY(7), /* LOOP CONTROL */
DESTADDR CHARACTER(1), /* DEST ADDRESS-USER INPUT */
/* GLOBAL VARIABLES */
TRMODE FIXED BINARY(7) EXTERNAL, /* VAX CMD FLAG */
VTERM FIXED BINARY(7) EXTERNAL, /* TERMINAL FLAG */
FRSIZE FIXED BINARY(15) EXTERNAL, /* FRAME SIZE */
/* GLOBAL DATA STRUCTURES */
TXBUFF(1508) FIXED BINARY(7) EXTERNAL; /* TRANS BUFF */

/*LAST REVISION: 09/15/83-0900 ORIGINAL PROGRAM:07/29/83*/
/*AUTHOR: CAPT. MARK D. STOTZER-USMC-AEGIS GROUP */
/*THESIS ADVISER: PROF. UNO R. KODRES-COMPUTER SCIENCE */

IF VTERM= 1 THEN /* TERMINAL MODE */
DO;
   CALL SENDMSG;
   RETURN;
END;
IF TRMODE= 1 THEN /* VAX COMMAND MODE */
DO;
   CALL SENDMSG;
   RETURN;
END;
COUNT5A=1;
DO WHILE (COUNT5A=1);
   PUT SKIP LIST('ADDRESSES ON THIS NETWORK:');
   PUT SKIP LIST('00-03-EA: MDS SYSTEM = 1');
   PUT SKIP LIST('00-04-0A: MDS SYSTEM = 2');
   PUT SKIP LIST('00-07-7F: VAX 11/790 = 3');
   PUT SKIP LIST('**************');
   PUT SKIP LIST('ENTER SELECTION ==>');
   GET LIST(DESTADDR);
   PUT SKIP(2);
   TXBUFF(1)=2; /*LOAD THE FIRST FOUR DEST ADDR BYTES*/
   TXBUFF(2)=7;
   TXBUFF(3)=1;
   TXBUFF(4)=3;
...
IF DESTADDR='1' THEN
  DO;
    TXBUFF(5)=3;/*LOAD LAST TWO DEST ADDR BYTES*/
    TXBUFF(6)=234;
    IF TXBUFF(7)=0 THEN/* SEND THE MSG*/
      CALL SENDMSG;
    ELSE
      CALL SENDFILE;/*SEND THE FILE*/
      COUNT5A=2;
  END;
ELSE
  IF DESTADDR='2' THEN
    DO;
      TXBUFF(5)=4;/*LOAD LAST TWO DESTINATION ADDR BYTES*/
      TXBUFF(6)=10;
      IF TXBUFF(7)=0 THEN
        CALL SENDMSG;
      ELSE
        CALL SENDFILE;
        COUNT5A=2;
    END;
  ELSE
    IF DESTADDR='3' THEN
      DO;
        TXBUFF(5)=7;/*LOAD LAST TWO ADDR BYTES*/
        TXBUFF(6)=127;
        TRMODE=0;
        IF TXBUFF(7)=0 THEN
          CALL SENDMSG;
        ELSE
          CALL SENDFILE;
          COUNT5A=2;
      END;
    ELSE
      PUT SKIP LIST('INVALID NET ADDRESS SELECTED-REENTER:');
      END; /* DO LOOP */
SENDMSG: /*MESSAGE SENDING MODULE*/
PROCEDURE;
DECLARE /* LOCAL VARIABLES */
  /* GLOBAL VARIABLES */
  FRSIZE FIXED BINARY(15) EXTERNAL,/*FRAME SIZE*/
  TRMODE FIXED BINARY(7) EXTERNAL,/*VAX CMD FLAG*/
  TTERM FIXED BINARY(7) EXTERNAL,/*TERMINAL FLAG*/
  /*GLOBAL DATA STRUCTURES*/
  TXBUFF(1508) FIXED BINARY(7) EXTERNAL,/*TRANS.BUFFER*/
  RXBUFF(1522) FIXED BINARY(7) EXTERNAL,/*RECV.BUFFER*/
  /*EXTERNAL MODULES*/
  FLDUF ENTRY,/*LOADS TRANS.BUFFER FROM CONSOLE*/
SENDFRAM ENTRY; /* ISO LEVEL 3 FRAME SENDER*/

IF VTERM=1 THEN /* VIRTUAL TERMINAL MODE */
  DO;
    CALL FILBUF;
    IF TXBUFF(9)=96 THEN
      RETURN;
    IF TXBUFF(9)=46 & TXBUFF(10)=96 THEN /*END SESSION*/
      VTERM=0; /*END TERMINAL SESSION*/
  ELSE
    CALL SENDFRAM;
  END;
ELSE
  DO;
    PUT SKIP LIST('MESSAGE SENDER: `');
    PUT SKIP LIST('MAXIMUM NUMBER OF CHARACTERS = `');
    PUT LIST(FRSIZE);
    PUT SKIP LIST('ENTER MESSAGE AFTER PROMPT: `');
    PUT SKIP LIST('END MESSAGE WITH ACCENT: `');
    PUT SKIP LIST('`');
    CALL FILBUF; /*FILL TRANSMIT BUFFER FROM CONSOLE*/
    CALL SENDFRAM; /* SEND THE MESSAGE */
  END;
END SENDMSG;

SENDFILE: /* FILE SENDING MODULE*/

PROCEDURE;
DECLARE /* LOCAL VARIABLES */
  COUNT4 FIXED BINARY(7), /*LOOP CONTROL*/
  /* GLOBAL VARIABLES */
  FILTYP FIXED BINARY(7) EXTERNAL, /*FILE NATURE*/
  FNOP FIXED BINARY(7) EXTERNAL, /*NOT OPEN FLAG*/
  LFRM FIXED BINARY(?) EXTERNAL, /*LAST DATA FLAG*/
  /* GLOBAL DATA STRUCTURES */
  TXBUFF(1508) FIXED BINARY(?) EXTERNAL,
  /* EXTERNAL MODULES */
  VAXTXT ENTRY, /*CP/M TO VAX FORMAT CONVERTER*/
  TRNDMA ENTRY, /*TRANSMIT SET DMA ADDRESS*/
  OPENDF ENTRY, /*OPEN DISK FILE*/
  RDISK ENTRY, /*READ DISK FILE RECORD*/
  SENDFRAM ENTRY; /*ISO LEVEL 3 FRAME SENDER*/

/*LAST REVISION: 08/25/83-1530 ORIGINAL PROGRAM:08/16/83 */
/*AUTHOR: CAPT. MARK D. STOTZER-USMC-AEGIS GROUP */
/*THESIS ADVISOR: PROF. UNO R. KODRES-COMPUTER SCIENCE */

TXBUFF(7)=15; /* LOAD TYPE FIELD BYTES*/
TXBUFF(8)=0;
CALL OPENDF;
IF FNOP=1 THEN /*FILE NOT ON DISK*/

71
DO;
  PUT SKIP LIST('FILE NOT ON DISK-REENTER DATA:');
  PUT SKIP(2);
  RETURN;
END;

IF TXBUFF(6)=127 & FILTYP=1 THEN
  CALL VAXTXT; /*VAX TEXT FILE FORMAT CONVERTER*/
ELSE
  DO;
    CALL TRNDMA; /*SET DISK DMA ADDRESS*/
    PUT SKIP LIST('****** FILE TRANSFER BEGINS *****');
    PUT SKIP(2);
    COUNT4=1;
    DO WHILE(COUNT4=1);
      CALL FDISK; /*READ A DISK FILE RECORD*/
      IF LFRM=1 THEN
        DO;
          CALL SENDFRAM;
          TXBUFF(8)=1; /*ENCODE TYPE FLD=INTERMED FRAME*/
        END;
      ELSE
        COUNT4=2;
      END; /*DO LOOP*/
      TXBUFF(8)=255; /*ENCODE TYPE FIELD=LAST FRAME*/
      CALL SENDFRAM;
    END;
  END; /*DO*/
  PUT SKIP LIST('****** FILE TRANSFER ENDS *****');
  PUT SKIP(2);
  RETURN;
END;
END SENDFILE;

END SENDATA; /*ISO LAYER 6 TRANSMIT MODULE*/
APPENDIX J

SOURCE CODE FOR MODULE RECDATA.PLL

RECDATA: /* ISO LAYER 6 RECEIVE MODULE */

PROCEDURE;

DECLARE /* GLOBAL DATA STRUCTURES */
    RXBUFF(1522) FIXED BINARY(7) EXTERNAL;/*RCV BUFF*/

/*LAST REVISION: 09/15/83-1215 ORIGINAL PROGRAM:08/17/83 */
/*AUTHOR: CAPT MARK D. STOTZER-USMC-AEGIS GROUP */
/*THESIS ADVISOR: PROF. UNO R. KODRES-COMPUTER SCIENCE */

IF RXBUFF(17)= 0 THEN /* MESSAGE FRAME */
    CALL CONMSG;
ELSE IF PXBUFF(17)= 15 THEN /* FILE FRAME */
    CALL FILER;
ELSE
    PUT SKIP LIST('RECEIVED IMPROPERLY ENCODED FRAME');

CONMSG: /* MESSAGE RECEIPT MODULE */

PROCEDURE;

DECLARE /* GLOBAL VARIABLES */
    TRMODE FIXED BINARY(7) EXTERNAL,/*VAX CMD FLAG*/
    FRSIZE FIXED BINARY(15) EXTERNAL,/*FRAME SIZE*/
    VTERM FIXED BINARY(7) EXTERNAL,/*TERMINAL FLAG*/
    /* GLOBAL DATA STRUCTURES */
    RXBUFF(1522) FIXED BINARY(7) EXTERNAL,/*RECV BUF*/
    /* EXTERNAL MODULES */
    TRMSG ENTRY,/* ACKNOWLEDGE SENDER*/
    EMTBUF ENTRY;/* DUMPS RECEIVE BUFFER TO CONSOLE*/

    IF VTERM=1 THEN /* NOT IN VIRTUAL TERMINAL MODE*/
        DC;
        PUT SKIP LIST('***** RECEIVED MESSAGE IS:');
        PUT SKIP(2);
        END;
        CALL EMTBUF; /* DUMP THE RECEIVED FRAME DATA TO CONSOLE */
        CALL TRMSG; /* SEND THE ACK FRAME */
    IF VTERM=1 THEN /* NOT IN TERMINAL MODE*/

73
DO:
    PUT SKIP(2);
    PUT SKIP LIST(('****** END OF MESSAGE TEXT.'));
    PUT SKIP(2);
    PUT SKIP LIST(PACK IN WAIT LOOP-ENTER<CR> TO EXIT=>');
    PUT SKIP LIST(('****************************'));
    PUT SKIP(2);
END;
ELSE
    IF RXBUFF(18)=15 THEN /*LAST FRAME OF TERMINAL REPLY*/
        PUT SKIP LIST(('V>'));
END CONMSG;

FILER: /* FILE FRAME RECEIPT MODULE*/

PROCEDURE;

DECLARE /* GLOBAL VARIABLES */
    TRMODT FIXED BINARY(7) EXTERNAL,/*CMD FLAG*/
    RECFIL FIXED BINARY(?),/*RFILE NO.*/
    VTERM FIXED BINARY(7) EXTERNAL,/*TERM FLAG*/
    /* GLOBAL DATA STRUCTURES */
    RXFCB EXTERNAL,/*RECEIVE FILE CONTROL BLOCK*/
    2 DISK FIXED BINARY(7),
    2 FNAME CHARACTER(8),
    2 FTYPE CHARACTER(3),
    2 TFCB(24) FIXED BINARY(?),
    RXBUFF(1522) FIXED BINARY(?),/*RX BUF*/
    /* EXTERNAL MODULES */
    RCVDMA ENTRY,/*SETS RECEIVE DISK DMA ADDR*/
    DELEDF ENTRY,/*DELETES FILES*/
    MAKEDF ENTRY,/*MAKES NEW DISK FILES*/
    WRDISK ENTRY,/*WRITES A DISK RECORD*/
    TRMSG ENTRY,/*SENDS ACK FRAMES*/
    CLOSDF ENTRY,/*CLOSES DISK FILES*/
CALL RCVDMA;
    IF RXBUFF(18)=0 THEN /*FIRST FILE FRAME*/
    DO:
        PUT SKIP LIST(('****** FILE RECEIPT BEGINS ******'));
        PUT SKIP LIST(OPENING FILE- RECFROM_NET:');
        PUT SKIP(2);
        RXFCB.FNAME='RECFROM_NET'; /*NAME THE RECEIVED FILE*/
        RXFCB.FTYPE='NET';
        RXFCB.TFCB(1)=0; /*ZERO THREE FIELDS OF FCB*/
        RXFCB.TFCB(4)=0;
        RXFCB.TFCB(21)=0;
        CALL DELEDF; /*DELETE OLD FILE OF THIS FN. FT*/
        CALL MAKEDF; /*CREATE A NEW ONE*/
        CALL WRDISK; /*WRITE FIRST RECORD(128 BYTES) TO DISK*/
    END:
CALL TRMSG; /* SEND THE FIRST ACK FRAME */
END;
ELSE
IF RXBUFF(18)=1 THEN /*INTERMEDIATE FILE FRAME*/
  DO;
    CALL WRDISK; /*WRITE NEXT RECORD TO DISK*/
    CALL TRMSG; /* SEND THE ACK FRAME */
  END;
ELSE
IF RXBUFF(18)=255 THEN /*LAST(DUMMY) FILE FRAME*/
  DO;
    CALL CLOSDF; /*CLOSE THE DISK FILE*/
    PUT SKIP LIST(‘******* END FILE RECEIPT *******’);
    PUT SKIP LIST(‘SEE FILE(S):RECFROM_NET’);
    PUT SKIP(2);
    CALL TRMSG; /*SEND THE LAST ACK */
    PUT SKIP LIST(‘---------- NOTE:’);
    PUT SKIP LIST(‘-----------------------------’);
    PUT SKIP LIST(‘IF RECEIVED FILE IS A TEXT FILE FROM’);
    PUT SKIP LIST(‘THE VAX THEN REFORMAT USING:’);
    PUT SKIP LIST(‘PIP FNAME.FTYPE=RECFROM_NET[D80]’);
    PUT SKIP LIST(‘WHEPT FNAME.FTYPE IS YOUR CHOICE’);
    PUT SKIP LIST(‘-----------------------------’);
    PUT SKIP(2);
    IF VTERM=1 THEN
      DO;
        PUT SKIP LIST(‘STILL IN VAX TERMINAL MODE:’);
        PUT SKIP LIST(‘V’);
      END;
    ELSE
      DO;
        PUT SKIP LIST(‘IN WAIT LOOP-ENTER<CR> TO EXIT’);
        PUT SKIP LIST(‘*********************************’);
        PUT SKIP(2);
      END;
    END;
ELSE
  PUT SKIP LIST(‘FRAME TYPE FIELD BYTE 2 INVALID CODE’);
END FILER;
END RECDATA; /* ISO LAYER 6 RECEIVE MODULE */
APPENDIX K

SOURCE CODE FOR MODULE ETHER2.ASM

;******************************************************************************
;******************************************************************************
; PROGRAM NAME: ETHER2.ASM
; THIS MODULE PERFORMS THE ISO LAYER 2 AND 3 FUNCTIONS IN
; TRANSMIT AND RECEIVE AND PROVIDES THE ISO LAYER 7
; RECEIVE MODULE

APPLICATION LAYER(LAYER 7): IN RECEIVE ONLY - WAIT LOOP
FOR FRAME ARRIVAL.

NETWORK LAYER(LAYER 3): TRANSMIT OR RECEIVE FRAMES

DATA LINK LAYER(LAYER 2): PROCESSES ACKNOWLEDGE FRAMES
IN ADDITION TO THE LAYER 2 FUNCTIONS PERFORMED BY THE
N13010 CONTROLLER BOARD.

THIS MODULE ALSO ALLOWS ALL OTHER MODULES TO ACCESS
THE CP/M-80 OPERATING SYSTEM FUNCTIONS SHOWN BELOW

LAST REVISION: 09/16/83-1000 ORIGINAL PROGRAM: 08/14/83
AUTHOR: CAPT MARK D. STOTZER-USMC-AEGIS MODELING GROUP
THESIS ADVISOR: PROFESSOR UNO R. KODRES-COMPUTER SCIENCE

;******************************************************************************
PUBLIC INIT; SUBROUTINES AVAILABLE TO EXTERNAL MODULES:
PUBLIC RECEIVE
PUBLIC FILBUF
PUBLIC EMDBUF
PUBLIC NULBUF
PUBLIC AWAIT
PUBLIC TRMSG
PUBLIC WRDISK
PUBLIC VAXTXT
PUBLIC SENDFRAM
PUBLIC RDISK
PUBLIC OPENDF
PUBLIC DELEDF
PUBLIC MADEDF
PUBLIC CIOSDF
PUBLIC FCVDMA; MODULES CALLED BY THIS MODULE

76
PUBLIC TRNDMA
EXTERN RECDATA

; NI3010 BOARD REGISTER PORT ADDRESSES:
CREG EQU 00B0H; COMMAND REGISTER
SREG EQU 00B1H; COMMAND STATUS REGISTER
ISREG EQU 00B5H; INTERRUPT STATUS REGISTER
IEREG EQU 00B8H; INTERRUPT ENABLE REGISTER
EBAR EQU 04B0H; EXTENDED BASE ADDRESS REGISTER
H3BAR EQU 04B3H; HIGH BASE ADDRESS REGISTER
LBAR EQU 04B2H; LOW BASE ADDRESS REGISTER
HBREG EQU 00BCH; HIGH BYTE COUNT REGISTER
LBREG EQU 00B9H; LOW BYTE COUNT REGISTER

;CP/M WARM BOOT ENTRY POINT:
EXIT EQU 0003H; WARM BOOT-TERMINAL ERROR ESCAPE

;BDOS EQUATES:
BDOS EQU 0605H; BDOS ENTRY POINT

;BDOS FUNCTION CODES:
CONIN EQU 01H; CONSOLE CHARACTER INPUT
CONSOUT EQU 02H; CONSOLE CHARACTER OUTPUT
PSTRING EQU 09H; PRINT STRING
CONSTAT EQU 0BH; CHECK CONSOLE STATUS
OPENFIL EQU 0FH; OPEN A DISK FILE
CLOSEF EQU 10H; CLOSE A DISK FILE
DELETE EQU 13H; DELETE A DISK FILE
READF EQU 14H; READ A DISK FILE RECORD-128 BYTES
WRITEF EQU 15H; WRITE A DISK FILE RECORD-128 BYTES
MAKEF EQU 16H; CREATE A NEW DISK FILE
SDMA EQU 1AH; SET DISK DMA ADDRESS

;INIT- Initializes INTERRUPT VECTOR AND NI3010 REGISTERS:

INIT DI
IN SREG; READ STATUS REGISTER TO CLEAR
MVI A,03FH; CLEAR NI3010 RECEIVE BUFFER
OUT CREG
CALL READ
MVI A,12H; SET UP INTERRUPT CONTROL
OUT 0FDH
MVI A,00H
OUT 0FCH
MVI A,0DFH; ENABLE INT5 ONLY
OUT 0FCH
MVI A,0C3H
STA 0028H
LXI H,RECFRAM
SHLD 0229H
LXI H,ACK
MVI A,0FFH; PRELOAD ACKNOWLEDGE BUFFER
MOV M,A
LXI H,CEREG; ENABLE RECEIVE(3BA) INTERRUPT
MVI A,04H
MOV M,A
OUT IREG
MVI A,09H; NI3010 ONLINE COMMAND
OUT CREG
CALL READ
EI RET

; RECALL ISO LAYER 7 WAIT LOOP FOR INCOMING FRAMES:
RECEIVE EI
WAITLP NOP
NOP
NOP
NOP
NOP
DI
LXI H,FRAMIN
MOV A,M
CPI 01H; HAS A FRAME ARRIVED?
JNZ NOTTET
CALL RECDATA
MVI A,00H; RESET FRAME ARRIVAL FLAG
STA FRAMIN
NOTTET MVI C,CONSTAT
CALL EDOS
CPI 00H
RNZ EI
JMP WAITLP

; RECFRAM—PERFORMS ISO LEVEL 3 FUNCTION IN THE RECEIVE
; MODE: RECEIVES FRAMES AND TRANSFERS THEM TO MEMORY.
; HANDLES ALL NI3010 INTERRUPTS AND ENABLES.
RECFRAM DI
PUSH PSW
PUSH 3
PUSH D
PUSH H
LXI H,CREG
MOV B,M
MVI A,00H
LXI H,CREG; DISABLE NI3010 INTERRUPTS
MOV M,A
OUT IREG
MVI A,3
MVI B,04H

78
CMP B, 07H
JZ RBA; RECEIVE FRAME INT WAS ENABLED
MVI A, 08H
CMP B
JZ RDD; RECEIVE DMA INT WAS ENABLED
JMP RDD2; IF TRANSMIT DMA INT WAS ENABLED
MVI A, 00H
OUT EBAR
LXI H, RBUFFT; TOP OF RECEIVE BUFFER
MOV A, H
OUT HBREG
MOV A, L
OUT LBREG
LHLDR FRSIZE
LXI D, 0016H; ADD 22 TO IT
DAD D
MOV A, H
OUT HBREG
MOV A, L
OUT LBREG
LXI H, CEREG
MVI A, 07H; SET INT ENABLE TO RDD
MOV M, A
OUT IEREG
JMP FINI

RDD
LXI H, RBUFFT; TOP OF RECEIVE BUFFER
MOV A, M
CPI 00H; TESTS FOR GOOD FRAME
JNZ FRERR; BAD RECEIVED FRAME
MVI A, 01H; SET FRAME ARRIVED FLAG
STA FRAMIN
LXI H, RTYPE1; TEST FOR RECEIVED ACK FRAME
MOV A, M
CPI 00H
JNZ RDD2
LXI H, RTYPE2
MOV A, M
CPI 0FFH
JNZ RDD2
MVI A, 01H
STA ACK; ACK FRAME RECEIVED
JMP RDD2

FRERR
DI H, CEREG
MVI A, 00H
MOV M, A; DISABLE BOARD INTERRUPTS
CUT IEREG
LXI D, FERMSG0
CALL TXTOUT
LXI D, TERRMSG
CALL TXTOUT
79
RDCP: CONTINUE TO READ THE BUFFER

RDCP
CPI 06H; BACKSPACE=8=CNTL-H
JZ BACKSP
CPI 02H; GRAVE ACCENT='=END OF MESSAGE
JZ SENT
MOV M,A; STORE THE CHAR.

FILBUF-PLACES CONSOLE INPUT MESSAGES INTO TRANSMIT BUFFER:

FILBUF
LHLD FRSIZE; LOAD COUNT=FRAME SIZE
XCHG
PUSH D
LXI H,TFDATA; LOAD ADDR =TRANSMIT DATA TOP
PUSH H
MVI C,CONIN; INPUT CONSOLE CHAR.
CALL BDOS
POP H
POP D
CPI 0DH; WAS CARRIAGE RETURN INPUT?
JNZ RDCP
PUSH H ; YES
LXI H,VTERM; IN TERMINAL MODE?
MOV A,M
CPI 01H
JZ VTEND; THEN THIS IS END OF MSG.
POP H
MOV M,A; STORE THE CHAR.
INX H
MVI A,0AH; ADD A LINE FEED
MOV M,A; STORE THE LINEFEED TOO
PUSH D
PUSH H
MVI C,CONSOUT; OUTPUT IT TO CONSOLE
CALL BDOS
POP H
POP D
JMP RDCON; CONTINUE TO READ THE BUFFER
DCX          D; DECREMENT THE COUNTER
MOV          A,D
ORA          E
JNL          RDCON; IF CTR NOT ZERO THEN CONTINUE READ
PUSH         H
LXI          D;LONGMSG;ERROR MSG:TOO MANY INPUT CHAR.
CALL         TXTOUT
VTEND        POP         H; TERMINAL MSG IN BUFFER-DONE
            MVI         A,60H
            JMP         SENT
RDCON        PUSH        D; CONTINUE BRANCH
            INX         H
            PUSH        H
            JMP         MSGLP; GET ANOTHER CHAR
BACKSP       INX         D
            PUSH        D
            DCX         H
            PUSH        H
            JMP         MSGLP; GET ANOTHER CHAR
SENT         MOV         M,A; STORE THE CHAR
            PUSH        H
            LXI         D,DADDI; LAST ADDR BYTE
            MOV         A,M
            CPI         0?FH; IS VAX =DESTINATION?
            JZ          SENFIN
            POP         H
            MVI         A,00H
            MOV         M,A; SOTRE A NULL IN PLACE OF ACCENT
            CALL        ECLN
            RET
SENFIN       POP         H
            CALL        ECLN
            RET

;***********************************************************************
;***********************************************************************
; EMTBUF-DUMPS RECEIVE BUFFER TO CONSOLE:
;***********************************************************************
;***********************************************************************
EMTBUF       LHL         FRSIZE
XCHG         D
LXI          H,RDATAT; TOP OF RECEIVE BUFFER
CONLP        MVI         C,CONSOUT; CHAR TO CONSOLE
            MOV         E,M
            PUSH        H
            CALL        BDOS
            POP         H
            POP         D
            DCX         D
            MOV         A,D
            ORA         E
            JZ          "MSGDONE; IF COUNT=FRAME SIZE-DONE"
VAXTXT—CONVERTS CPM FORMAT TEXT FILES TO VAX FORMAT:

VAXTXT

CALL EOLN
MVI C,OPENFIL; OPEN THE DISK FILE
LXI D,PCBIN
CALL BDOS
CPI OFFH; TEST IF OPEN SUCCESSFUL
JZ FERR1
MVI C,SDMA; SET THE DISK DMA ADDRESS
LXI D,TXTTOP
CALL EOLN
LXI H,TXTTOP; TOP OF TEXT BUFFER
PUSH H
LXI D,TFDATA; TRANSMIT BUFFER 1ST DATA BYTE
PUSH D
MVI 3,00H; BYTE CTR=0
PUSH B
CALL NULBUF; FILL TRANSMIT BUFFER WITH 00 HEX

READREC

MVI C,READF; READ A DISK FILE RECORD=128 BYTES
LXI D,PCBIN
CALL BDOS
CPI 00H; IS THIS LAST RECORD?
JNZ ENDRD
RDLPA

POP B
POP D
POP H
INR 3; INCREMENT COUNTER
MOV A,B
CPI 081E;=129 LAST BYTE THIS RECORD
JZ READ2; GET ANOTHER RECORD
MOV A,M
CPI 0DH; CRET?
JZ SKIP2
CPI 0AH; LFEED?
JZ SKIP3
XCHG
MOV M,A
XCHG
INX H
INX D
PUSH H
PUSH D
PUSH B
JMP RDLPA
INX H; IF BYTE=CRET THEN SEND THE FRAME
PUSH H
LXI D,TFDATA
PUSH D
PUSH B
CALL SENDFRAM; SEND IT
CALL NULBUF; NULL THE BUFFER AGAIN
MVI A,01H; SET TYPE FIELD=INTERMED FRAME
STA TTYP2
JMP RDLPA; READ NEXT BYTE AFTER SKIP CRET
SKIP2
INX H; IF LINEFEED THEN SKIP AND READ MORE
PUSH H
PUSH D
PUSH B
JMP RDLPA
READ2
LXI H,TXTTOP; IF CTR >128 THEN GET RECORD
PUSH H
INX D
PUSH D
MVI B,00H; RESET BYTE CTR
PUSH B
JMP READREC GET THE NEXT RECORD
SKIP3
ENDRD
MVI A,0FFH
STA TTYP2
POP B
POP D
POP H
CALL SENDFRAM
LXI D,DMSG
CALL TXTOUT
RET ; DONE
FERR1
LXI D,ERMSG; ERROR MSG-FILE NOT OPEN
CALL TXTOUT
RET

; ISO LEVEL 3 TRANSMIT FUNCTION-SENDFRAM:
; SENDFRAM- SENDS FRAMES ON THE ETHERNET:
;
SENDFRAM DI
LOCP1 LXI H,CEREG; LOOP UNTIL ENABLE REG= 2 OR 4
MCV A,M
CPI 00H
JZ GO
CPI 04H
JZ GO

83
EI
JMP LOOP1: KEEP CHECKING
GO
DI
LXI H,CEREG
MOV A.M
CPI Ø0H
JZ GO1
CPI Ø4H
JZ GO1
EI
JMP LOOP1: IF CHANGED GO BACK TO LOOP
GO1
MVI A,Ø0H
LXI H,CEREG; DISABLE NI3010 INTERRUPTS
MOV M,A
OUT IEREG
EI
MVI A,Ø0H; LOAD TRANSMIT ADDR/BYTE COUNT
OUT EBAR
LXI H,TBUFFT; TOP OF TRANSMIT BUFFER
MOV A,H
OUT HBREG
MOV A,L
OUT LBREG
DI
MVI A,Ø6H
LXI H,CEREG; ENABLE TRANSMIT(TDD) INTERRUPT
MOV M,A
OUT IEREG
ELT; WAIT FOR THE INTERRUPT
COMP
LXI H,CEREG
MOV A,M
CPI Ø6H; HAS TDD INTERRUPT ARRIVED?
JZ COMP
DI
LXI H,VTERM
MOV A,M
CPI Ø1H; VIRTUAL TERMINAL MODE?
JZ VTCON
LXI D,MSG1
CALL TXTOUT
VTCON
EI
MVI A,Ø29H; NI3010 LOAD TRANSMIT AND SEND CMD.
DI
OUT CREG
CALL TRREAD
LXI H,ACK; SET ACK TO SENT
MVI A,00H
MOV M,A
EI
CALL AWAIT; WAIT FOR ACKNOWLEDGE FRAME
RET

; ISO LEVEL 2 ROUTINES: AWAIT(TRANSMIT) AND TRMSG(RECEIVE);

; AWAIT—WAITS FOR RETURN OF ACKNOWLEDGE FRAMES:

AWAIT LXI D,0000FH; FIRST TIMER LOOP COUNTER
TRNLPS LXI B,0FFFFH; INNER LOOP
TRNLPS1 LXI H,ACK
MOV A,M
CPI 01H; RECEIVED ACK YET?
JZ BACK
DCX B
MOV A,C
ORA B
JNZ TRNLPS
DCX D
MOV A,E
ORA D
JNZ TRNLPS1
CALL TXTOUT
LXI D,TIMMSG; TIMED OUT—ABORT
CALL TXTOUT
LXI D,TERREMSG
CALL TXTOUT
JMP EXIT; ESCAPE TO CPM
MVI A,0FFH; RESET ACK FLAG
STA ACK
MVI A,00H; RESET FRAME ARRIVAL FLAG
STA FRAMIN
RET

; TRMSG—SENTS ACKNOWLEDGE FRAMES IN RECEIVE MODE:

TRMSG MVI C,03H; CTR=3
LXI H,SRCADDD
LXI D,DADDD
LOOP2 MOV A,M
XCHG MOV M,A
XCHG DCR C
JZ LDCONT
INX
INX
JMP LOOP2
MVI A,02EH; RESET INTERRUPT PRIORITY
OUT OFDH
MVI A,00H
OUT EBAR
LXI H,TBUFFT
MOV A,H
OUT EBAR
MOV A,L
OUT LBAR
LELD FRIZE
LXI D,0000H
DAD D
MOV A,H
OUT HRREG
MOV A,L
OUT LRREG
MVI A,00H; LOAD TYPE FIELD=ACK FRAME
STA TTYP1
MVI A,0FFH; ACK FRAME
STA TTYP2
MVI A,06H; ENABLE TDD INTERRUPT
LXI H,CEREG
OUT MLREG
EI
HIT ; WAIT FOR THE INTERRUPT
DONE
LXI H,CEREG
MOV A,M
CPI 06H; TRANSMIT DMA DONE?
JZ DONE
DI
MVI A,029H; LOAD TRANSMIT AND SEND COMMAND
OUT CREG
CALL THREAD
RET

; OPERATING SYSTEM SUBROUTINES:
;
RDISK MVI A,00E; READS A DISK FILE RECORD=128 BYTES
STA LFRM ; PRELOAD LAST FRAME FLAG
LXI D,PCBIN
MVI C,READF
CALL EDOS
CPI 00H; =NOT LAST FRAME
RZ
MVI A,01H; =LAST FRAME
STA LFRM
; WRDISK  MVI  C,%WRIT; WRITES DISK FILE RECORD-128 BYTES
LXI  D,FCBOUT
CALL  BDOS
CPI  00H
JNI  DWERR
LXI  D,WRMSG
CALL  TXTOUT
RET

; DWERR  LXI  H,CEREG
MVI  A,00H; DISABLE BOARD INTERRUPTS
OUT  IEREG
LXI  D,DRVMSG
CALL  TXTOUT
JMP  EXIT; ESCAPE TO CPM

; OPENDF  MVI  A,07H; OPENS DISK FILES
STA  FNOP
LXI  D,FCBIN
MVI  C,OPENFIL
CALL  BDOS
CPI  0FPH; OPENING ERROR
RNZ
MVI  A,01H
STA  FNOP
RET

; DELEDF  LXI  H,RECFIL; DELETES EXISTING DISK FILES
MOV  A,M
INR  A; INCREMENT RECEIVED FILE NUMBER
STA  RECFIL
STA  FCBOUT+8
LXI  D,FCBOUT
MVI  C,DELETE
CALL  BDOS
RET

; MAKEDF  LXI  D,FBCOUT; MAKES A NEW DISK FILE
MVI  C,MAKFF
CALL  BDOS
RET

; CLOSDF  LXI  D,FBCOUT; CLOSES A DISK FILE
MVI  C,CLOSEF
CALL  BDOS
RET

; RCVDMA  LXI  D,RDATAT; SETS DISK DMA FOR RECEIVE MODE
MVI  C,SDMA
CALL  BDOS
RET
TRNDMA LXI D,TFDATA; SETS DISK DMA ADDR FOR TRANSMIT
MVI C,SDMA
CALL BDOS
RET

UTILITY SUBROUTINES:
READ—READS THE COMMAND STATUS REGISTER AFTER EACH COMMAND:

READ MVI B,11111110B
MVI C,0Eh
STLP IN ISREG
ORA B
CPI 0FFH; STATUS READY TO BE READ?
JNZ STLP
IN SREG
CPI C
JZ STDONE
JMP ERROR

STREAD MVI B,11111110P
STLP1 IN ISREG
ORA B
CPI 0FFH
JNZ STLP1
IN SREG
CPI 00H
JZ STDONE
CPI 01H
JZ STDONE
ERROR LXI D,EMSG
CALL TXTOUT
STDONE RET

TXTOUT—OUTPUTS TEXT STRINGS TO THE CONSOLE:

TXTOUT MVI C,PSTRING
CALL BDOS
CALL EOLN
RET

EOLN—GENERATES CARRIAGE RETURN + LINE FEED:

EOLN MVI C,CONSOUT
MVI E,0DH
CALL BDOS
MVI C,CONSOUT
MVI E,0AH
CALL BDOS
RET
; NULBUF—FILLS THE TRANSMIT BUFFER WITH NULLS (00 HEX):
;
NULBUF MVI C,0000H; CTR=128
LXI H,TFDATA
NULLOOP MVI A,00H
MOV M,A
DCR C
RZ
INX H
JMP NULLoop

; STORAGE ALLOCATION:

; NEEDED MESSAGES:

TRMSGI DB "**** FILE TRANSFER BEGINS ****$
DMSG DB "**** FILE TRANSFER COMPLETE *****$
ERMSG DB "FILE NOT ON DISK$"
NORESMSG DB "ON RESPONSE FROM VAX-EXITING TO CPMS$"
LONGMSG DB "MAX CHARACTER LENGTH REACHED-MSG SENTS$
TERMSG DB "UNRECOVERABLE ERROR-EXITING TO CP/M$"
TIMMSG DB "TIMED OUT-ABORTING TRANSMISSION$"
EMSG DB "N13010 COMMAND FAILED$
MSG1 DB "TX$"
FERMSG DB "RECEIVED BAD FRAME$"
WRMSG DB "RX$"
DWMSG DB "DISK WRITE ERROR-DISK FULLS"
COMMON/TXFCB/ FCBIN DS 36; TRANSMIT FILE CONTROL BLOCK
COMMON/RXFCB/ FCBOUT DS 36; RECEIVE FILE CONTROL BLOCK
COMMON/TXBUFY/ TBUYFT DS 1; TRANSMIT BUFFER TOP-1ST DEST ADDBYTE
DADDB DS 1; SECOND DEST ADDR BYTE
DADDC DS 1; THIRD DEST ADDR BYTE
DADD DS 1; FOURTH DEST ADDR BYTE
DADDRE DS 1; FIFTH DEST ADDR BYTE
DADDF DS 1; SIXTH DEST ADDR BYTE
RTYPE1 DS 1; FIRST RECV TYPE FLD BYTE
RTYPE2 DS 1; SECOND RECV TYPE FLD BYTE
TFDATA DS 1500; DATA FIELD MAX SIZE
COMMON/RXBUFF/ RBUF7T DS 13; RECEIVE BUFFER TOP-FRAME CHECK BYTE
SRCADDD DS 1; FOURTH SRCE ADDR BYTE
SRCADDE DS 1; FIFTH SRCE ADDR BYTE
SRCADDF DS 1; LAST SRCE ADDR BYTE
RTYPE1 DS 1; FIRST RECVD FRAME TYPE FLD BYTE
RTYPE2 DS 1; SECOND RECVD TYPE FLD BYTE
RDATAT DS 1500; RECEIVED DATA FIELD MAX SIZE
CRCBYT DS 4; CRC FIELD
COMMON/TXTBUP/ TITTOP DS 128; VAX TEXT TEMP BUFFER
COMMON/FRSIZE/ FRSIZE DS 2; ACTUAL FRAME DATA BLOCK SIZE
COMMON/ACK/ ACK DS 1; ACKNOWLEDGE FLAG LOCATION
COMMON/FNOP/ FNOP DS 1; FILE NOT OPEN FLAG
COMMON/LFRM/ LFRM DS 1; LAST FRAME FLAG
COMMON/TRMODE/ TRMODE DS 1; VAX TRANSMIT FLAG
COMMON/FILTYP/ FILTYP DS 1; TYPE OF FILE TO SEND
COMMON/RECFIL/ RECFIL DS 1; RECEIVED FILE NUMBER
COMMON/VTERM/ VTERM DS 1; VIRTUAL TERMINAL SERVICE FLAG

; END: ASSEMBLY LANGUAGE MODULE ETHER2.ASM
;******************************************************************************
;******************************************************************************
;******************************************************************************
APPENDIX L

TEST PROGRAM USER INSTRUCTIONS

The Ethernet hardware test programs, ETHTESTA and ETHTESTB, are used in the manner below:

1. Invoke either program using normal CP/M-80 procedures.

2. Both programs first command the NI3010 to run its built-in diagnostic tests and report failures to the user via the console. The codes that ETHTESTB will display as ASCII letters are encoded as noted at the end of the ETHTESTB.ASM source listing.

3. Next, both programs ask the user to input a short line of text that the programs use in testing the integrity of the essential data paths of the NI3010. Program ETHTESTB will ask the user for a second text line input because it performs one more test than ETHTESTA. The maximum number of characters per line is 42 and the line must be ended with a grave accent: "".

4. The tests are successful if no error indications are displayed on the console and the text typed in is shown on the console exactly as it was entered after each data path input.
The instructions for use of the communication program ETHERNET.COM are as listed below:

1. Invoke the program ETHERNET using normal CP/M-80 procedures.

2. The program will then ask for the selection of:
   A. The disk drive number to write any received files to.
   B. The desired number of data bytes per Ethernet frame.
   C. The network service desired. The choices are:
      1. Send messages or files.
      2. Receive messages or files.
      3. Virtual terminal service with the VAX.
      4. Command file transfers to or from the VAX.
      5. Disconnect from the network.

Depending on which of the above services is requested by the user, the program will do the following:

1. Send a file or message: The program will ask the user to specify which one and, depending on the response, will do the following:
   A. If message sending is selected, the program will:
      1. Ask the user to choose the network address of the destination.
      2. Then ask the user to input the message itself. The maximum message size is determined by the previously selected data block size. The last character entered in order to transmit must be a grave accent character: "\".
      3. The message is then sent and upon successful receipt by the destination host the program restarts.
   B. If file sending is selected, the program will:
      1. Ask the user if the file is a text or machine code file.
      2. Ask the user to specify which disk the file is located on.
      3. Ask the user the filename and filetype of the
file.

4. Ask the user to specify the network address of the destination.

5. Upon successful transmission of the entire file the program will restart.

2. Receive a file or message: The program will, upon selection of this mode, wait in a loop for any transmissions addressed to it to arrive. After the receipt of any file or message, the program will return to the wait loop. This feature allows the user to leave the system unattended and then send multiple files and/or messages to it from another network host. The program numbers files in the order they are received beginning with RECFROM0.NET, etc. Text files received from the VAX must be run through the CP/M PIP utility as follows:

   "PIP newfilename.filetype=RECFROM0.NET[DE0]" which will chop off unneeded characters. The user can exit the wait loop to return to the above menus by entering a carriage return.

3. Terminal service with the VAX 11/780: The program will display a set of instructions to the user concerning the operation of the program in this mode. The user can input text after each V-prompt (V>) appearance. To exit this mode, the user must enter a period (.) followed by a carriage return immediately following any V-prompt (V>). Upon exiting this mode, the program returns to the beginning user menus.

4. Command VAX file transfers: This mode allows the INTELLEC system to command the VAX to either send or receive files by sending it specially coded messages. The procedure is as follows:

   A. Downloading VAX files:
      1. The user must enter the message:
         "1VAX filename.VAXfiletype/TXT or EXE"'
      2. The specified VAX file will then be sent to the requesting unit.
      3. In the above message, TXT refers to text and EXE refers to machine code files.
      4. After the file receipt is completed, the user can exit the wait loop by entering a carriage return.

   B. Uploading VAX files:
      1. The user must enter the message:
         "2VAX filename.VAXfiletype/TXT or EXE"
      2. The above message opens a file by the above filename and filetype on the VAX. The VAX will reply: "Ready for sendfile FN.FT' and the
program will be in the receive wait loop.

3. The user must then enter a carriage return to the beginning of the program and then follow the normal file sending procedures as noted above.

5. Disconnect from the network: Selection of this mode causes the program to return control to the CP/M-80 operating system.

The other features of this program are as follows:

1. Error handling: The below listed transmission or reception errors will cause the program to display error messages and return to CP/M-80:
   A. Receipt of a bad frame.
   B. Receipt of a frame that has an improperly encoded type field.
   C. Acknowledge frame not received by the sending host in a given time frame (Source timed out).
   D. Receipt of a file larger than the disk space remaining (Disk full).

2. Special instructions for IAPX 432 files that must be transferred from the VAX to an INTELLEC system running the Intel ISIS-II operating system:
   A. These special files can only be transferred using the VAX comma mode. The VAX/VMS program ETHERNET.EXE must be invoked on the VAX in order for this transfer to be successful.

3. The procedure is as follows:
   1. After downloading the file to the INTELLEC double density system using ETHERNET.COM and CP/M-80, the user must rename it from the name assigned to it by the receive program to its original name.
   2. The user must put the CP/M-80 disk in drive A which must have stored on it both the renamed file and the program TOISIS.COM.
   3. The user must then insert an ISIS-II disk into drive B.
   4. The user then, while logged on drive A, must invoke TOISIS filename.filetype. This will convert the program on disk A to the ISIS-II format and store it on disk B.
   5. The user must then remove the CP/M-80 disk in drive A and replace it with the disk from drive B.
   6. The last step is to reboot the INTELLEC system under the ISIS-II operating system and proceed with the IAPX 432 procedures.


BIBLIOGRAPHY


Miller, A. M., Mastering CP/M, Sybex, 1983.

1. Defense Technical Information Center  
   Cameron Station  
   Alexandria, Virginia  22314

2. Library, Code 0142  
   Naval Postgraduate School  
   Monterey, California  93943

3. Department Chairman, Code 62  
   Department of Electrical Engineering  
   Naval Postgraduate School  
   Monterey, California  93943

4. Department Chairman, Code 52  
   Department of Computer Science  
   Naval Postgraduate School  
   Monterey, California  93943

5. Professor Uno R. Kodres, Code 52Kr  
   Department of Computer Science  
   Naval Postgraduate School  
   Monterey, California  93943

6. Professor Mitchell L. Cotton, Code 62Cn  
   Department of Electrical Engineering  
   Naval Postgraduate School  
   Monterey, California  93943

7. Professor Alex Gerba, Jr., Code 62Gz  
   Department of Electrical Engineering  
   Naval Postgraduate School  
   Monterey, California  93943

8. LtCol. Alan Ross, USAF, Code 52Rs  
   Department of Computer Science  
   Naval Postgraduate School  
   Monterey, California  93943

   Department of Computer Science  
   Naval Postgraduate School  
   Monterey, California  93943
10. Capt. Mark D. Stotzer, USMC  
    12802 Greenhall Drive  
    Woodbridge, Virginia  22192  

11. LtCol. J.F. Mullane, USMC, Code 0309  
    United States Marine Corps Representative  
    Naval Postgraduate School  
    Monterey, California  93943  

12. First Lieutenant Thawip Netniyom, RTA  
    Chulachomkloc Royal Military Academy  
    Rajadamnurn Avenue  
    Bangkok, Thailand  

13. Capt. Ted F. Rogers, USN  
    Box 327  
    Lumberport, West Virginia  26386  

14. Captain Ioannis A. Karadimitropoulos  
    Delvinou 16  
    Papagou  
    Athens, Hellas  

15. Mr. Roger H. Stotzer  
    Langston Incorporated  
    111 Woodcrest Road  
    Cherry Hill, New Jersey  08034  

16. Lieutenant Ioannis K. Kidoniefs, Hellenic Navy  
    SMC 2303  
    Naval Postgraduate School  
    Monterey, California  93943  

17. Major Anthony K. Sakellaropoulos, Hellenic Air Force  
    SMC 2243  
    Naval Postgraduate School  
    Monterey, California  93943  

18. Mr. Mike Williams, Code 52  
    Department of Computer Science  
    Naval Postgraduate School  
    Monterey, California  93943  

19. Daniel Green (Code N20E)  
    Naval Surface Warfare Center  
    Dahlgren, Virginia  22449
20. Cdr. J. Donegan, USN
PMS 40035
Naval Sea Systems Command
Washington, D. C. 20362

21. Mike McGowan
3585 198 Avenue
Aloha, Oregon 97007

22. Dr. M. J. Gralia
Applied Physics Laboratory
Johns Hopkins Road
Laurel, Maryland 20707

23. Dana Small
Code 8242
NOSC, San Diego, California 92152