

MICROCOPY RESOLUTION TEST CHART  
NATIONAL BUREAU OF STANDARDS-1963-A

AD-A164 076



DEVELOPMENT AND IMPLEMENTATION  
 OF THE  
 X.25 PROTOCOL  
 FOR THE  
 UNIVERSAL NETWORK INTERFACE DEVICE (UNID) II  
 VOL I OF II

THESIS

AFIT/GE/ENG/85L-52

Mark W. Weber  
 Captain

USAF

DTIC FILE COPY

DTIC  
 FEB 13 1986

DEPARTMENT OF THE AIR FORCE  
 AIR UNIVERSITY

**AIR FORCE INSTITUTE OF TECHNOLOGY**

Wright-Patterson Air Force Base, Ohio

This document has been approved  
 for public release and sale by the  
 distribution statement.

86 2 12 09

DEVELOPMENT AND IMPLEMENTATION  
OF THE  
X.25 PROTOCOL  
FOR THE  
UNIVERSAL NETWORK INTERFACE DEVICE (UNID) II  
VOL I OF II

THESIS

AFIT/GE/ENG/85D-52

Mark W. Weber  
Captain

USAF

**S** DTIC  
ELECTE **D**  
FEB 13 1986  
**E**

DEVELOPMENT AND IMPLEMENTATION  
OF THE  
X.25 PROTOCOL  
FOR THE  
UNIVERSAL NETWORK INTERFACE DEVICE (UNID) II  
VOL I OF II  
  
THESIS

Presented to the Faculty of the School of Engineering  
of the Air Force Institute of Technology  
Air University  
in Partial Fulfillment of the  
Requirements for the Degree of  
Master of Science in Electrical Engineering

Mark W. Weber, BS EE  
Captain, USAF

December 1985

<b>Accession For</b>	
NTIS GRA&I	<input checked="" type="checkbox"/>
DTIC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By _____	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-1	



## PREFACE

This research effort describes the continued development of an improved Universal Network Interface Device (UNID II). The UNID II's architecture was based on a preliminary design project at the Air Force Institute of Technology. The UNID II contains two hardware modules; a local module for the network layer software and a network module for the data link layer software and physical layer interface. Each module is an independent single board computer (SBC) residing on an Intel multibus chassis, complete with its own memory (EPROM and RAM), serial link interfaces, and multibus interface. The local module is an iSBC 544 and the network module is an iSBC 88/45. This research effort expands the Consultive Committee for Telephone and Telegraph (CCITT) X.25 protocol in the UNID II design. This report updates the documentation for the detailed hardware and software design, test, and integration of this system.

I wish to thank thank all the people who helped me through this challenging project. First, there is my thesis advisor, Dr. Gary Lamont, for guidance and advice during my periods of confusion. My thanks as well to Major Walter Seward and Dr. Hartrum for their assistance and guidance. I greatly appreciate the assistance from Mr. Orville Wright, Mr. Charlie Powers, Mr. Dan Zambalm, and Mr. Robert Durham for their technical and supply support. Capt Ken Cole gets a special thanks for pointing me in the right direction an many different occasions. Finally I reserve my highest thanks to thank my wife, Melissa, for her patience and never ending support given during these 18 months.

Table of Contents

	<u>Page</u>
Preface . . . . .	ii
Table of Contents . . . . .	iii
List of Figures . . . . .	vii
List of Tables. . . . .	ix
Abstract. . . . .	x
I. Introduction and Background. . . . .	1-1
Background. . . . .	1
Current Status. . . . .	8
Problem Statement . . . . .	9
Scope . . . . .	9
Assumptions . . . . .	10
Summary of Current Knowledge. . . . .	10
Standards . . . . .	10
Approach. . . . .	11
Equipment . . . . .	12
Other Support . . . . .	12
Conclusion. . . . .	13
II. UNID II and DELNET Requirements . . . . .	2-1
Introduction. . . . .	1
UNID II Requirements Summary. . . . .	1
DELNET Functional Requirements. . . . .	8
X.25, Network Layer Protocol. . . . .	10
Test and Validation Requirements. . . . .	12
Protocols . . . . .	14
Implementation Issues in the ISO Model . . . . .	15
Conclusion. . . . .	19
III. System Design . . . . .	3-1
Introduction. . . . .	1
Synopsis. . . . .	2
ISO Reference Model . . . . .	3
Transport Layer . . . . .	6
Internet Protocol . . . . .	9
Subnet. . . . .	12
Network Layer . . . . .	13
Data Link Layer . . . . .	15
Physical Layer. . . . .	19
Conclusion. . . . .	21

Table of Contents

	<u>Page</u>
IV. UNID II Hardware Design . . . . .	4-1
Introduction. . . . .	1
Initial Hardware Design . . . . .	1
Childress's Design. . . . .	7
Detailed Hardware Description of the SBC 544 (56) . . . . .	10
Detailed Hardware Description of the SBC 88/45 (55) . . . . .	11
SBC 86/12A Hardware Description (54). . . . .	14
AM 95/6445 Card Cage Description. . . . .	14
UNID II Physical Layer Interface. . . . .	15
Conclusion. . . . .	17
V. Software Design. . . . .	5-1
Introduction. . . . .	1
Previous Development. . . . .	1
Development Language Selection. . . . .	13
UNID II Data Structures . . . . .	15
Network 3B Design . . . . .	16
Network 3A Design . . . . .	25
Data Link Design and Implementation . . . . .	26
Conclusion. . . . .	40
VI. Software Integration and Validation . . . . .	6-1
Introduction. . . . .	1
Test Philosophy . . . . .	1
Integration/Validation. . . . .	2
Test Outline. . . . .	5
Phase One Testing . . . . .	6
Phase Two Testing . . . . .	7
Phase Three Testing . . . . .	16
Phase Four Testing. . . . .	21
Conclusion. . . . .	23
VII. Conclusions and Recommendations. . . . .	7-1
Introduction. . . . .	1
Conclusions . . . . .	1
Recommendations . . . . .	2
Concluding Remarks. . . . .	6
Bibliography. . . . .	BIB-1



Table of Contents (cont)

	<u>Page</u>
Appendix A. UNID II Data Flow Diagrams. . . . .	A-1
UNID II Data Flow Diagrams (31:26-34) . . . . .	1
Appendix B. RS-232C and RS-422 Signals. . . . .	B-1
RS-232C and RS-422 Signals (15:Appendix B). . . . .	1
Appendix C. Hardware Configuration for the UNID II. . . . .	C-1
Hardware Configuration for the UNID II. . . . .	1
Appendix D. Host System Modifications for the UNID II . . . . .	D-1
Host System Modifications for the UNID II . . . . .	1
Appendix E. UNID Semaphores and Protected Regions . . . . .	E-1
UNID Semaphores and Protected Regions (15:Appendix E) . . . . .	1
Appendix F. Transmit Request/Transmit Acknowledge Handshake . . . . .	F-1
Transmit Request/Transmit Acknowledge Handshake (15:Appendix F) . . . . .	1
Appendix G. DELNET/UNID Header Information. . . . .	G-1
DELNET/UNID Header Information (15:Appendix D). . . . .	1
Appendix H. UNID II Software Data Dictionary. . . . .	H-1
Data Dictionary (15:Appendix G) . . . . .	1
1. Network Layer Simulation . . . . .	2
Constants . . . . .	2
Variables . . . . .	3
Procedures. . . . .	5
Link and Locate Batch File. . . . .	6
2. Data Link Layer Simulation . . . . .	7
Constants . . . . .	7
Variables . . . . .	8
Procedures. . . . .	10
Link and Locate Batch File. . . . .	11

Table of Contents (cont)

	<u>Page</u>
3. SBC 544 Validation . . . . .	12
Constants . . . . .	12
Variables . . . . .	14
Link and Locate Batch File. . . . .	15
4. Host CP/M Simulation . . . . .	16
Constants . . . . .	16
Variables . . . . .	16
Procedures. . . . .	18
Link and Locate Batch File. . . . .	19
Appendix I. UNID II Software Structure Charts . . . . .	I-1
ISIS HOST . . . . .	2
Operational SBC 544 Software. . . . .	7
Data Link Simulation Software . . . . .	16
Appendix J. ISIS HOST Software. . . . .	J-1
ISIS HOST Software . . . . .	1
Appendix K. Host I/O Modules. . . . .	K-1
1. INTEL 210 CP/M I/O Module (assembly). . . . .	2
2. INTEL 230 ISIS I/O Module (assembly). . . . .	11
3. INTEL 230 ISIS I/O Module (PL/M) . . . . .	15
Appendix L. SBC 544 Simulation Software . . . . .	L-1
SBC 544 Simulation Software . . . . .	1
Appendix M. SBC 544 Operational Software. . . . .	M-1
SBC 544 Operational Software . . . . .	1
Include file INTR3.LOC . . . . .	37

Table of Contents (cont)

	<u>Page</u>
Appendix N. Data Link Simulation Software . . . . .	N-1
1. Main Module. . . . .	2
2. LAPBO Module . . . . .	45
3. LAPB1 Module . . . . .	84
4. PCKT Module. . . . .	97
Vita. . . . .	V-1

List of Figures

<u>Figure</u>		<u>Page</u>
1 - 1	Figure 1-1. Initial Concept of a Multi-Ring Base Level Network (87) . . . . .	1-2
1 - 2	Figure 1-2. DELNET Architecture (87) . . . . .	7
2 - 1	Figure 2-1. UNID II General Concept. . . . .	2-3
2 - 3	Figure 2-2. UNID II, Country Code "0", General Concept. . . . .	4
2 - 3	Approximate Correspondence Between the Various Networks (105:22). . . . .	16
2 - 4	INA 960 Conceptual Diagram (65:27) . . . . .	18
3 - 1	Figure 3-1. Temporal Order Sublayering in CCITT X.25 Recommendation (3:22). . . . .	3-4
3 - 2	Figure 3-2. ISO OSI Reference Model Applied to DELNET and UNID(15:2-7) . . . . .	4
3 - 3	Figure 2-3. Transport Header used with DELNET (78:C-20 - C-25) . . . . .	9
3 - 4	Figure 3-4. ISO OSI Reference Model with the Internet Protocol (15:2-9). . . . .	11
3 - 5	Internet Protocol used in DELNET (78:C-20 - C-25). . . . .	12
3 - 6	Network Header Defined by CCITT (108:25) . . . . .	15
3 - 7	Data Link Frame Format Defined by CCITT (108:25) . . . . .	17
3 - 8	Previously Implemented Frame Header Information (15:2-11). . . . .	17
3 - 9	Currently Implemented Frame Header Information . . . . .	18
4 - 1	Figure 4-1. UNID II Block Diagram (30:43). . . . .	4-3
4 - 2	Figure 4-2. UNID II Block Diagram (Revised) (64:1-9) . . . . .	6
4 - 3	Figure 4-3. Current UNID II Block Diagram. . . . .	8
4 - 4	Figure 4-4. A RS-422 Inter-connection Technique for UNID II. . . . .	15
4 - 5	The Implemented RS-422 UNID Interconnection Technique. . . . .	16
5 - 1	Figure 5-1. Original UNID Data Structures and Flow (15:4-4) . . . . .	5-3
5 - 2	Figure 5-2. UNID II Data Structures and Flow . . . . .	4
5 - 3	Figure 5-3. Network Layer High Level Structure Chart (15:4-8) . . . . .	7
5 - 4	Figure 5-4. Route\$In Procedure Structure Chart (15:4-8) . . . . .	7
5 - 5	Figure 5-5. Route\$Out Procedure Structure Chart (15:4-12). . . . .	8
5 - 6	Figure 5-6. Route\$Out Procedure Pseudocode (15:4-12) . . . . .	9
5 - 7	Figure 5-7. UNID/Host Transmit Request/Transmit Acknowledge Handshake (15:4-13). . . . .	10
5 - 8	Figure 5-8. UNID II Memory Map (15:4-30) . . . . .	12

List of Figures (cont)

<u>Figure</u>	<u>Page</u>
5 - 9 Figure 5-9. Control Lead Sequence for Half Duplex Operation (27:956) . . . . .	19
5 - 10 Figure 5-10. Control Lead Sequence for Full Duplex Operation (27:955). . . . .	20
5 - 11 Figure 5-11. Pseudocode for a full Duplex DTE . . . . .	22
5 - 12 Figure 5-12. Transmit Interrupt Procedure (15:4-17)	23
5 - 13 Figure 5-13. Route\$out Procedure Pseudocode . . . . .	24
5 - 14 Figure 5-14. Pseudocode for Main Procedure Data Link Software. . . . .	28
5 - 15 Pseudocode for Procedure START\$DM\$MODE. . . . .	29
5 - 16 Pseudocode for Procedure START\$INFO\$XFER. . . . .	29
5 - 17 Pseudocode for ROUTE\$IN Data Link Procedure . . . . .	31
5 - 18 ROUTE\$IN Destination Processing Pseudocode. . . . .	32
5 - 19 Pseudocode for RCV\$I\$FRAME. . . . .	35
5 - 20 Example of a Processed I Frame. . . . .	37
6 - 1 Figure 6-1. Network Layer Simulation Data Structure and Flow (15:5-7). . . . .	6-10
6 - 2 Figure 6-2. Data Link Layer Simulation Data Structure and Flow (15:5-13) . . . . .	12
6 - 3 Pseudocode for for Procedure READLINE. . . . .	13
6 - 4 Frame Delay Test . . . . .	19
6 - 5 Figure 6-5. Network Layer Simulation with the CP/M System and H19 (15:5-13) . . . . .	19
6 - 6 Figure 5-4. UNID II and NETOS Connection (15:5-14) . . . . .	21
A - 1 Figure A-1. UNID II Overview . . . . .	A-2
A - 2 Figure A-2. Input Local Information. . . . .	3
A - 3 Figure A-3. Format According to Outgoing Protocol. . . . .	4
A - 4 Figure A-4. Transmit Network Message . . . . .	5
A - 5 Figure A-5. Input Network Information. . . . .	6
A - 6 Figure A-6. Transmit Local Information . . . . .	7
B - 1 Figure B-1. RS-232C Pin Assignments. . . . .	B-2
B - 2 Figure B-2. RS-422 Pin Assignments . . . . .	3
E - 1 Figure E-1. Pseudocode for SBC 544 to SBC 88/45 Packet Movement. . . . .	E-3
E - 2 Figure E-2. Pseudocode for SBC 88/45 to SBC 544 Packet Movement. . . . .	6
F - 1 Figure F-1. NETOS Transmit Request/Transmit Acknowledge Handshake. . . . .	F-2
F - 2 Figure F-2. UNID TR/TA Allowable States. . . . .	3
F - 3 Figure F-3. State Diagram of the TXTR Handshake. . . . .	4
G - 1 Figure G-1. DELNET/UNID Detailed Header Information. . . . .	G-5

List of Tables

<u>Table</u>			<u>Page</u>
Table 1 - 1.	Table 1.1	Phases of UNID Design . . . . .	1-4
Table 2 - 1.	Table 2-1	UNID II REquirements . . . . .	2-7
Table 2 - 2.	Table 2-2	UNID I and UNID II Validation Tests	2-13
Table 7 - 1.	Table 7-1	Tasks Accomplished . . . . .	7-1

## Abstract

This research effort describes the continued development of an improved Universal Network Interface Device (UNID II). The UNID II's architecture was based on a preliminary design project at the Air Force Institute of Technology. The UNID II contains two main hardware modules; a local module for the network layer software and a network module for the data link layer software and physical layer interface. Each module is an independent single board computer (SBC) residing on an Intel multibus chassis, complete with its own memory (EPROM and RAM), serial link interfaces, and multibus interface. The local module is an Intel iSBC 544 and the network module is an Intel iSBC 88/45. The network layer software supports the CCITT X.25, datagram option, protocol and the data link layer software supports the CCITT X.25 LAPB (HDLC) protocol. This report documents the further implementation of the CCITT X.25 protocol in the UNID II design.

## Chapter I

### Introduction and Background

This Thesis describes further development and implementation of the International Consultive Committee for Telegraph and Telephone (CCITT) X.25 protocol standard in the Universal Network Interface Device (UNID) II. This chapter provides introductory information and background material on UNID, UNID II, and the Digital Engineering Laboratory Network (DELNET). Background material for this thesis effort comes from material contained in the master's theses of Phister (93), Matheson (84) and Childress (15). Phister and Matheson detail the development and logical choice of the multi-ring structure chosen for UNID. Childress summarizes early UNID hardware and software development and presents the current UNID and DELNET status.

The following sections in this chapter give the background, problem statement, scope, current status, assumptions, standards approach, and equipment necessary to conclude the Thesis effort.

#### Background

Requirements for the UNID first came about in 1977 when the 1842 Electrical Engineering Group (EEG) identified the need of a local area network (LAN) to interconnect base-level data processing equipment. A report, "An Engineering Assessment Toward Economical, Feasible, and Responsive Base-Level Communications Through the 1980's (1), to the Air Force Communications Command (AFCC) concluded that high capacity communications could be achieved through the use of a multi-ring computer network structure. The multi-ring structure would meet typical base-



level requirements by providing flexibility, easy expansion, and high capacity at low costs. The original concept identified five different functional devices needed to interface various base data processing equipment. These five fundamental groups defined the services that the LAN would provide. Figure 1-1 shows the original multi-ring concept.

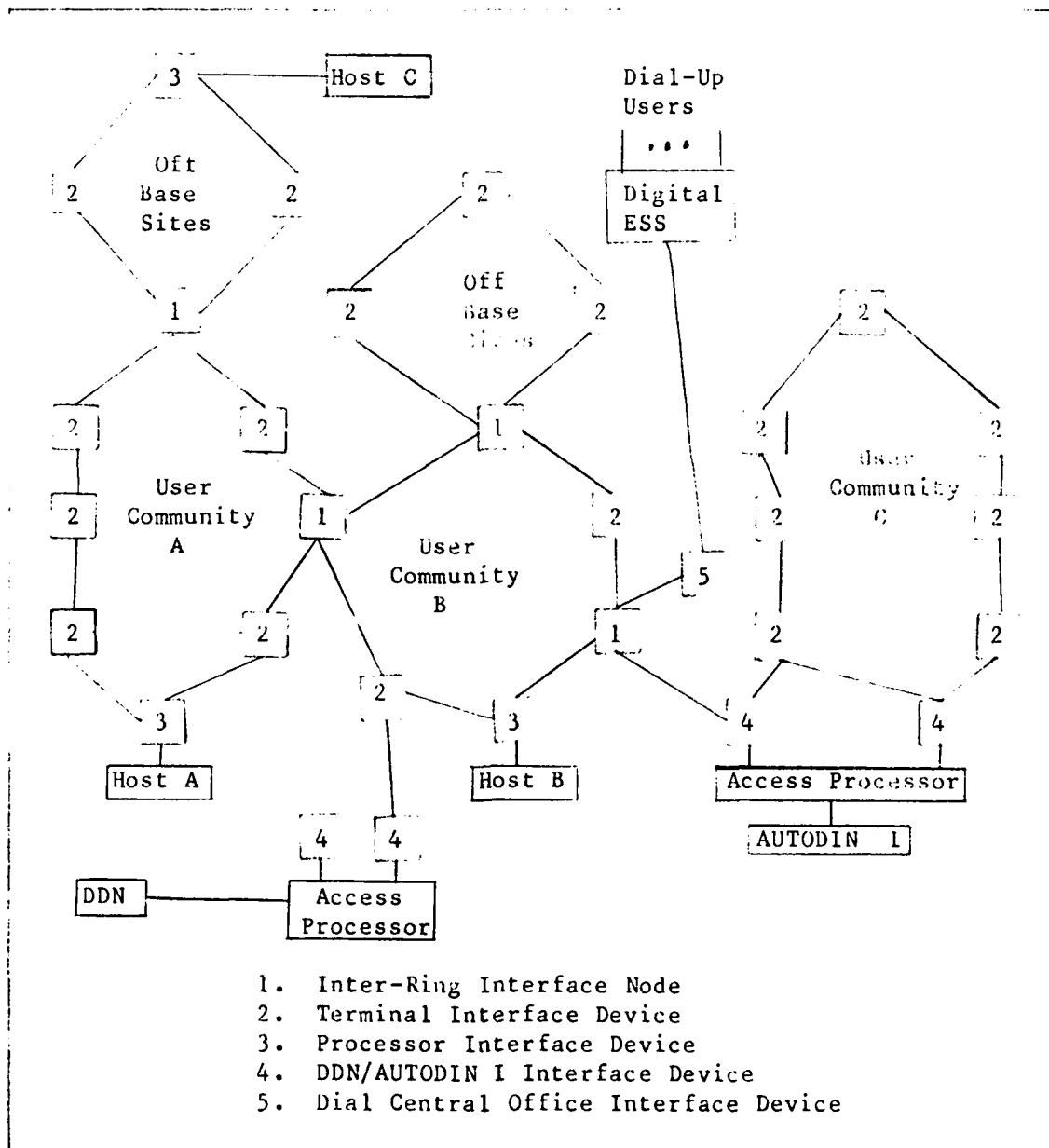


Figure 1-1. Initial Concept of a Multi-Ring Base Level Network (93)

Following the 1842 EEG's report, Rome Air Development Center (RADC) was tasked by AFCC to expand the multi-ring network concept and further define the required interface devices.

RADC's investigation of the functional interface resulted in a generalized interface device, performing all functions required by the multi-ring LAN. The device depicted by RADC's studies would connect a highly variable group of host processing equipment to form small closed rings and then connect the separate rings to form one large, base-wide multi-ring LAN. The name given to the device to accomplish all interconnections between host elements nodes on the multi-ring network was Universal Network Interface Device or UNID. According to Phister (93:1-4) the multi-ring network concept, and hence UNID, was incorporated into the RADC post doctoral study program. Several research efforts from AFIT came about through AFIT's association in RADC's Post Doctorial program.

The original 1842 EEG concept as modified by RADC prompted a series of AFIT thesis efforts that produced detailed studies of a multi-ring network environment and hardware implementing the findings of those studies. Past thesis efforts include the works of Sluzevick (107) and Ravenscroft (96) in 1978; Brown (10) in 1979; Baker (5) in 1980; Papp (93), Hobart (34), Gravin (31), and Geist (30) in 1981; Cuomo (19), Palmer (91), and Hazelton (33) in 1982; Spear (108), Phister (93), and Matheson (84) in 1983; and, most recently, Childress (15) in 1984. These interests at AFIT forged two directions. First, development proceeded on a device to meet the criteria established by RADC using Z-

80 based architecture. This device became known as UNID I. Second, further development proceeded using 16 bit 8086 based architecture. This device was called UNID II.

Overall, the plan was to develop evolutionary hardware and software designs of the UNID architecture. Ultimately, the UNID designs would be used inside DELNET for educational purposes associated with computer network courses and for continued research in LAN design. The first Master's level thesis efforts came in 1978 and are briefly discussed in Childress's Thesis (15). The following material comes mostly from the introductory material of Childress.

In 1978 Sluzevich (107) and Ravenscroft (96) began their efforts on conceptual hardware and software designs of UNID and DELNET respectively. Sluzevich defined four phases of the UNID design (15:1-4).

Table 1-1. Phases of UNID Design

X	1. Define the functional requirements of the UNID.
X	2. Translate functional requirements into system design.
X	3. Design UNID's hardware.
-	4. Design UNID's software.

X: Phase completed  
-: Phase incomplete

To date, only the last task remains uncompleted. The design envisioned by Sluzevich consisted of three separate modules: one module would interface the UNID with host equipment, a second module would interface

the UNID with the multi-ring LAN, the third module would provide necessary data processing and control for the other modules. Ravenscroft centered on design of a LAN for the Digital Engineering Laboratory (DEL). This network evolved into what is presently known as DELNET.

In the following year, Brown (10) expanded the processing capability of the basic UNID architecture by providing separate parallel processors for the local host side and the network side of UNID. To facilitate data transfer, the two processors would share common memory blocks. In 1980 Baker (5) developed more software, expanded the hardware capability of the existing UNID, and performed system intergration tests with commercial fiber optic communication links.

In 1981 Papp (97) implemented an operational hardware structure and began initial hardware testing of two UNIDs in a DELNET configuration. His work formed the basic hardware components of what became UNID I. Work continued on DELNET with Hobart's conceptual development of required software (34). His structured analysis and design techniques (SADTs) formed the initial data flow diagrams of the network. Also in 1981, Giest started protocol development on UNID. His protocol structure used the ISO Open Systems Interconnection (OSI) seven layer model as the basic structure for data flow and program development.

In 1982 Cuomo and Hazelton began their efforts with UNID and DELNET respectively. Cuomo (19) refined hardware design of UNID I by expanding data ports available to hosts on UNID, reducing transient noise through rewiring the UNID I, and refitting the UNID I with static Random Access Memory (RAM). Hazelton (33) improved Giest's protocol by implementing software according to the Consultive Committee International Telephone

and Telegraph (CCITT) recommendations. At the data link layer, he started implementation of the Link Access Protocol (LAP) and, at the network layer, he began implementation of the CCITT X.25 network layer protocol.

In 1983 Phister and Matheson continued their respective efforts on UNID and DELNET. Phister (93) continued development on the data link layer and network layer software for implementation of the X.25 protocol layer using the datagram option. He also started development of the transport layer software based on the CCITT X.121 and Transmission Control Protocol/Internet Protocol (TCP/IP) standards. At this point DELNET standards were established to interface UNID at the transport level. Phister's work was the last effort on the Z-80 based hardware design known as UNID I. Matheson's work centered on the Intel SBC 86/12A single board computer (SBC) which formed the basic design of UNID II. Figure 1-2 shows the DELNET architecture according to Phister's design.



Palmer's design by completing the local and network modules. Matheson then started the translation of software developed by Phister to the PLM 86 programming language for the 8086 based UNID II.

In 1984 Childress (15) implemented the host and network modules of UNID II with two off-the-shelf SBC's. The Intel SBC 544 was selected as the local module implementation and the Intel SBC 88/45 was selected as the network module implementation. Together, these two boards eliminated the requirement for a separate processing module and provided reliable, proven hardware components for the UNID II design. Childress then completed the translation of Phister's software into PLM 86 and tested the local host SBC 544 board software. While software was developed for the SBC 88/45 network board, the software was never embedded on the board and tested as with the SBC 544 local board. At this point Childress concluded his development of UNID II.

#### Current Status

A hardware design using Intel SBC 544 Intelligent Communications Controller and SBC 88/45 Advanced Data Communications Processor Board exists with software embedded on the SBC 544 board. Software for the SBC 88/45 board exists, but has only been tested in modular form and has not been embedded in Read Only Memory (ROM) on the SBC 88/45. Test programs exist on the Intel System Implementation Supervisor (ISIS) and on a CP/M based operating system. These simulation programs implement the minimal amounts of transport layer protocol necessary to validate accurate data transmission by UNID II. Implementation of the network layer protocol consists of the data communications state of the datagram option of the CCITT X.25 (1980) protocol standard (115:133). Publications since 1984

indicate CCITT will drop the datagram option (105:41). This has no significant impact in development of the X.25 protocol within UNID II since planned development eventually includes full implementation of X.25 protocol.

#### Problem Statement

The intent of this research effort is to expand the CCITT X.25 protocol service within UNID II by implementing the data link software on the SBC 88/45 and expand the existing datagram service.

More Specifically:

1. Remove the current timing problems within the UNID II interrupts.
2. Install and validate the data link layer software on the SBC 88/45 network board.
3. Further develop and implement CCITT X.25 network layer protocol standards for full datagram service and permanent virtual circuit service (PVC).
4. Research current implementations of the transport, session, presentation, and applications layers of the ISO seven layer model for DELNET and UNID application.

#### Scope

The SBC 88/45 and SBC 544 board architecture will be investigated for an alternative method of disabling interrupts on incoming packets. The five signalling lines of the RS-232C interface, not yet fully implemented, but necessary to emulate the X.21(bis) protocol, will be developed. The software loop on the data link layer will be used as a bias for developing the data link procedures. Ultimately the procedures will



then be installed on the SBC 88/45 board. The data link receive ready, receive not ready, reject, set asynchronous balanced mode, disconnect, unnumbered acknowledgement, and command reject response frames will be implemented. Finally, an investigation of commercially available software packages for the upper four layers of the ISO model (Transport through Applications layers) will be performed.

#### Assumptions

These three assumptions made come from Childress (15:1-9) and remain valid for this thesis:

1. It is assumed that the local and network boards work properly.
2. It is assumed that the local software design previously developed, translated, and implemented functions properly.
3. It is assumed that the network software design as developed functions properly.

#### Summary of Current Knowledge

Childress' Thesis describes the most current development of UNID and DELNET (15). Both Childress's Thesis and Phister's Thesis give detailed summaries of past development of UNID and DELNET. Phister's Thesis provides further aid in developing the Transport layer interworkings.

#### Standards

Standards for this Thesis effort continue unchanged from the works of Childress (15:1-10), Phister (93:1-23) and other past research efforts (10, 19, 31, 33, 34, 93, 94, 92, 96, 107, 108). As such, these standards contain:

1. ISO Open Systems Reference Model DP-7490

2. CCITT X.1, X.2, and X.95 for Class of Service
3. CCITT X.121 for routing control
4. Transmission Control Protocol/Internet Protocol (TCP/IP).
5. CCITT X.25 (1980) for network control
6. ISO 3309-1976(E) for data link control
7. High Level Data Link Control (HDLC) protocol
8. CCITT Link Access Control B (LAPB)
9. RS-232C, RS-422 and RS-449 for physical layer protocol

These standards reflect current international agreements for public data communications networks. Furthermore, these standards maintain compatibility with DELNET and UNID I. The compatibility within DELNET between UNID I and UNID II allows the UNID devices to perform their basic function as a truly "universal" interface device between networks and computer hosts.

#### Approach

First, the UNID's timing and interrupt response when processing multiple datagrams will be examined using the HP 4951A protocol analyzer. A simulated host will send multiple datagrams to UNID II. The HP 4951A can monitor activity on the physical, data link, and network layers within UNID. Changes required to the interrupt structure will be made at that time.

Second, examination of the physical layer protocol will determine changes necessary to implement the CCITT X.21(bis) protocol as its RS-232-C and RX-422 equivalent. This implementation will require the HP

4951A and Intel System III acting as hosts on the UNID II.

Third, software for the data link layer will be installed in the SBC 88/45. Should RS-232-C implementations functions require alterations of the data link software, the software will be installed and tested before any other alterations are made.

Fourth, with the data link software installed, remaining datagram functions and CCITT X.25 permanent virtual circuit packets will be developed and implemented. Software will be developed on the Intel Series III computer using ISIS software development tools.

Finally, dependent on the time remaining, software packages implementing transport and higher layer software functions will be investigated.

#### Equipment

Software program development and host simulation requires use of the Intel Series III micro computer. Further host emulation and protocol analysis requires use of the HP 4951A protocol analyzer. Word-processing and Programmable Read Only Memory (PROM) requirements will be satisfied by the Intel System II/210 micro computer and Bytek PROM programmer. PROM programming will be necessary for embedding software on Eraseable/Programmable Read Only Memory (EPROM) in the SBC 544 and SBC 88/45 boards.

#### Other Support

A workbench, desk, and technical manuals for all hardware and software development tools will be required. Also, specific EPROMS, ROMS, integrated circuits (ICs) cable harnesses, and miscellaneous administrative supplies will be required.

## Conclusion

Chapter I, the introduction and background, began with a brief presentation of the ordering of material contained within the main body of this Thesis. Background material followed which gave a brief history of the UNID and DELNET development. Chapter I presented brief summaries of the current status of UNID II, problem statement, scope, assumptions, standards, approach, equipment and other support. Chapter II gives the UNID II and DELNET requirements pertaining to this Thesis. After Chapter II, the system design, hardware design, software design, and validation efforts follow in succeeding chapters. Chapter III details the system design as viewed through the protocol. Chapter IV details the current hardware design of UNID II, and Chapter V presents the current software design of UNID II. Chapter VI gives the Design and Implementation of the X.25 protocol features implemented to date. Chapter VII presents the final conclusions and recommendations of the Thesis.

## Chapter II

### UNID II and DELNET Requirements

#### Introduction

This chapter summarizes UNID II and DELNET requirements. Though the requirements have changed little since initially conceived (31, 84, 93), the degree of their functional implementation has varied greatly. None of the requirements were changed by this thesis effort. It is the objective of this thesis to further the extent of the implementation and not research new or different requirements. As such, the presentation here comes mostly from the works of Gravin, Phister, Matheson, and Childress. First, the UNID II functional and hardware requirements are given. Following UNID requirements, DELNET short and long term requirements detail UNID II and DELNET interactions. Next, the protocol requirements for UNID and DELNET are discussed. Then the validation of software and hardware/software integration are reviewed. Finally, the requirements integrating OpenNet products into the UNID and DELNET environment are presented for completeness.

#### UNID II Requirements:

Sluzevich, in 1979 developed the following general criteria (107:14):

1. The UNID should function as a store and forward concentrator and have message routing capabilities.
2. The UNID might require specialized I/O ports for unique communications requirements.
3. The UNID should be capable of interfacing to various network operating systems and protocols.

4. The UNID should provide an environment for computer communications studies.

These criteria formed the preliminary design goals of past UNID II development and remain valid for current UNID II development. In the final form UNID II will be incorporated into DELNET, and, within DELNET, UNID II will act as a research tool for analyzing design characteristics within Local Area Networks (LANs). Functionally UNID II will operate as a gateway switch and local cluster network incorporated in one device. As a gateway switch, UNID II will perform necessary protocol conversion from a host to a high speed dual ring network formed by other UNIDs. Separate dual ring networks will be linked by special UNIDs having the country code '0' (93:2-9). As was shown in Chapter I, These UNIDs will form the multi-ring LAN. The local side of UNID performs much the same as a concentrator node. Each UNID II will have the software capability of addressing as many as 256 hosts. As currently implemented in hardware, each UNID II acts as a concentrator node for up to four hosts. The hardware allows for possible expansion of the number of hosts a single UNID II may support. Figure 2-1 shows the general concept of the UNID II operation. As shown UNID consists of the subnet or bottom three layers of the ISO Reference Model plus the internet protocol. The local side of a UNID II may address up to 255 hosts, while the network side of UNID II supports up to 16 UNIDs within a single dual ring network.

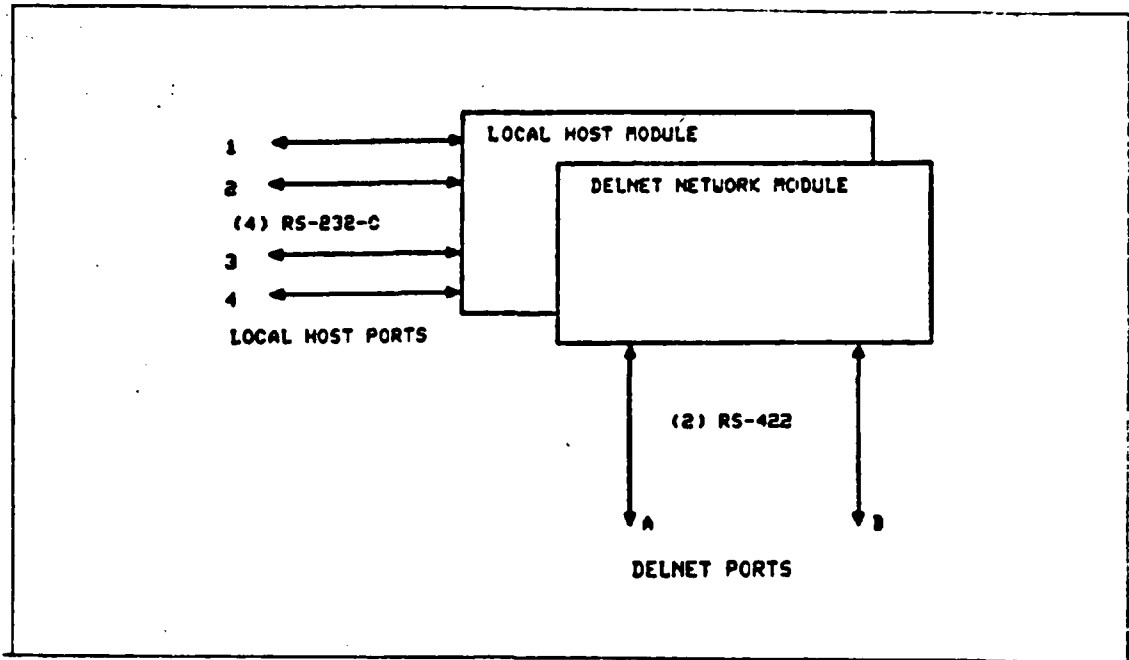


Figure 2-1. UNID II General Concept.

The UNIDs having a country code '0' have not been implemented in hardware. Figure 2-2 gives their general concept. As shown in Figure 2-2, a UNID II, country code '0' (UNID II (0)) is simply another UNID II with the addition of a second dual ring network interface. This gives UNID II (0) added control and responsibilities within the network. One of the features discussed by Phister was the use of UNID (0) to perform "wake up" programming of a ring of UNIDs upon activation of the network. Each non-UNID (0) (UNID (N) where N is any element of the set 1 to 15) member of ring would have capability of transmitting and receiving packets on the ring. Higher level functions would be contained within UNID (0) and upon activation of the network UNID (0) would program each member of the network. Thus all flow control, catastrophic failure actions, and normal network configuration information need

reside within only one UNID, the UNID (0) of each ring. The final requirements and specifications for UNID (0) have not yet been fully developed and remain as follow-on investigations for further UNID development (See Chapter VII for recommendations).

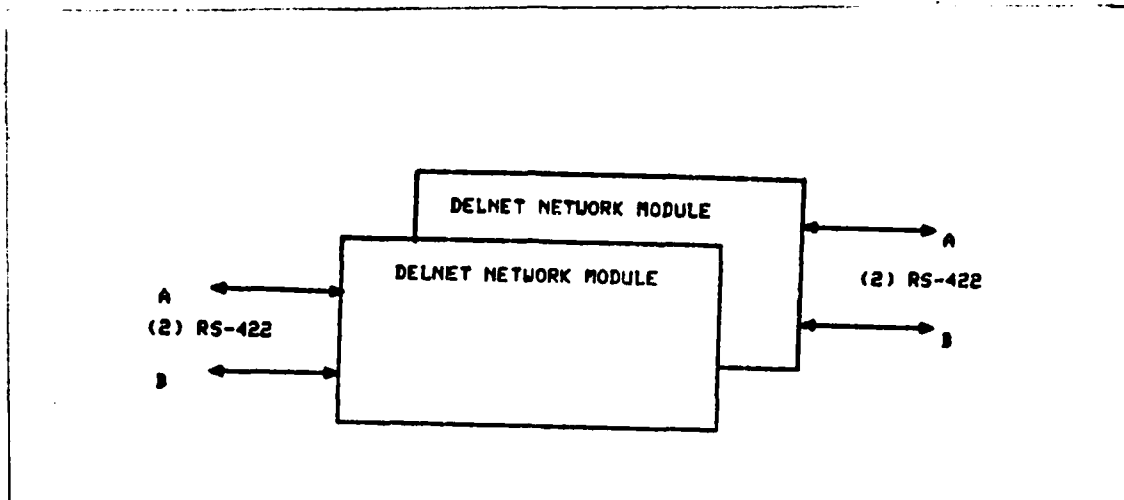


Figure 2-2. UNID II, Country Code '0' General Concept.

In defining the architecture of UNID II (0), consideration must be given to have all protocol changes in all UNIDs accomplished at the software or firmware level rather than changing or redesigning hardware. This will require hardware configured by software access within UNID. The following UNID II requirements summary supplies more details into the hardware requirements of UNID II and comes in part from Matheson (93:2-1 - 2-3).

Hardware requirements were developed from existing EIA standards for electronic interfaces. The electrical path of the local and network sides of UNID II forms the lowest level of the protocol implementation.



At this level, the physical level, UNID II will use RS-232-C (24) and RS-449 (25) standards. The characteristics of these two interfaces are covered in the physical layer discussion of the ISO Reference Model presented later within this chapter. The host to UNID local interface will use the RS-232-C interface as it is among the most common used by peripheral devices within the United States and conforms to higher level protocol requirements. UNID to UNID network interfaces will use the RS-449 interface. Upwardly compatible with the RS-232-C, the RS-449 interface allows the higher data rates required on the network side of the UNID. The RS-232-C and RS-449 interface protocols will enable the higher levels of protocol at the data link layer and above to be independent of the physical characteristics of the network. Therefore changes may be made at any of the higher levels without requiring design changes of the physical layer.

Sluzevich's structured analysis and design techniques (SADTs) (107) developed the original UNID design requirements. From these requirements, three separate modules were identified:

1. A local input/output module for interfacing the UNID to the user's computers, terminals, or modems.
2. A network module for interfacing the UNID to other UNIDs over the network.
3. A dual processor module for matching the local I/O to the network environment (107:154-155).

In 1981, Gravin (31) began design on an improved UNID based on 8086 components. From the original functional requirements (107:33) used for UNID I, data flow diagrams (DFDs) were used to produce a new functional requirements model for the UNID II. Table II-I lists those original requirements used to produce the UNID II. The DFDs arrived at a similar

design as did Sluzevich with one fundamental difference. Two functionally similar groups of requirements were identified similar to two of the modules identified by Sluzevich. One group handled messages between hosts on the local side of UNID. The second group handled messages between UNIDs in the network. The difference lay in that a third module was not identified by the DFDs. However, Matheson's design for UNID II continued Sluzevich's three module concept by partitioning the controlling facilities outside of the local and network modules. Childress, however, altered the design by implementing the intelligence within the local and network modules. A third processor module then was not required. The original DFDs developed by Gravin are reproduced in Appendix A. They were the basis for the UNID II design and remain valid for the current UNID II implementation.

TABLE 2-1: UNID II REQUIREMENTS (84)

- I. Interface a wide variety of network components and handle various topologies
  - A. Accommodate dissimilar computing equipment
    - 1. Accomplish code conversion
    - 2. Perform data rate speed conversion
  - B. Interface peripherals and user terminals to the network
  - C. Interface host computers to the network
  - D. Provide a network to network interface (a gateway)
- II. Perform independently of network components
  - A. Handle network data transmission and reception
    - 1. Accommodate network throughput requirements and flow control
    - 2. Adapt to different protocols
      - a) Handle both synchronous and asynchronous communication messages
      - c) Unpack a message
      - d) Perform parallel to serial and serial to parallel data conversion
      - e) Handle error control functions such as message acknowledge, no acknowledge, repeat and time out
    - 3. Perform error checking and recovery procedures
  - B. Relieve host computers from network specific functions
    - 1. Provide a buffer to smooth message traffic
    - 2. Poll communications lines if they are multidropped
    - 3. Handle interrupts
    - 4. Route messages to desired destinations
    - 5. Collect performance, traffic and error statistics
- III. Provide a test bed for computer network studies and research

### DELNET Requirements

The system requirements outlined in the original thesis (101) became the basis for a series of additional investigations. As the system requirements evolved, further refinement of the DFDs was accomplished (34), followed by implementing standards, developing software procedures and writing the necessary code (30, 33, 84, 93). At the same time the UNID hardware design was refined (5, 19, 84, 108) to correct known problems, improve reliability and create the dual ring topology of the UNID network (93). Demonstrations of the DELNET were accomplished in 1981 (92), 1983 (84), and 1984 (15) but were limited to the use of single UNID due to the lack of an operational second UNID and a complete software implementation of the network layer.

Hazelton in 1981 determined functional requirements for DELNET by performing a survey (33). Matheson summarized these requirements (84:2-3 - 2-6) and, as in Childress' work, they remain the basic requirements for DELNET. The requirements as summarized by Matheson are:

1. Ability to transfer files across the network.
2. Ability to share peripherals attached to the hosts on DELNET.
3. Flexibility with respect to the network topology, protocols, and transmission ports.
4. Performance monitoring capability.
5. High percentage of availability.
6. User transparency to network configuration and specific operating systems at a time.

While the above features state requirements for initial UNID functions, Hazelton questioned potential users for long term design goals. His results indicated long term UNID development would implement

functions to:

1. Permit software tool sharing.
2. Perform distributed processing.
3. Use distributed databases.
4. Incorporate fault tolerance.
5. Provide a means to connect to other networks such as ARPANET and the planned AFITNET.
6. Connect to the local Cyber and DEC VAX 11/780 mainframe computers.
7. Provide data privacy.
8. Provide security for classified projects.

In consideration of the last item, though very desirable, the DELNET and UNID are considered separate from systems implementing secure data communications. At present, no attempt will be made to develop the complex and detailed criteria required by the Electronic Security Command (ESC). Under existing conditions ESC will not approve a secure network in the AFIT environment due to: lack of physical security; lack of TEMPEST approved equipment; and lack of trusted computer software, and it's associated hardware, validated and approved by the National Security Agency (NSA). However, providing data privacy, including user authentication, and verification procedures for the DELNET users are desirable goals and possible within the scope of the DELNET and UNID. Recommendations in this area have been made since Hazelton's initial work and continue as recommendations for future thesis research (See Chapter VII for recommendations).

Utilizing the user generated list of functional requirements, a set of requirements for the DELNET hardware and software was established. A

ring topology was initially selected for the DELNET connections with each node providing a star subnet to the local users. This was the same basic configuration recommended in the 1842 EEG Technical Report (1) for base level communications systems. However, the report indicated a single user at each network interface while the DELNET requirement was for multiple hosts (up to four in the current design) on each network node. The chief advantages in using the ring topology was the development of easy routing algorithms and simplified system expansion (112:Chapter 7). The simple routing within the dual ring allows connection of new nodes to the network without changes in the routing algorithms.

In summary, the requirements for DELNET follow Hazelton's work and essentially remain unchanged. Phister has added details to the requirements by implementing the ISO Reference Model as it would be applied Hazelton's original DELNET requirements. Matheson and Childress both summarize these same requirements and, as such, the requirements for DELNET remain unchanged for this thesis as well.

#### X.25 Network Layer Protocol Requirements

In 1981, Giest (30) reviewed various protocol standards to determine the 'best' suited standards for DELNET. From his studies the CCITT X.25 packet switching recommendation was selected for establishing the protocol standards within DELNET. The criteria for 'best' sought to include long term durability and an international scope for the standard. These criteria have been achieved as demonstrated by the incorporation of the X.25 recommendation in the Institute of Electrical and Electronics Engineers (IEEE) LAN specifications (IEEE 802) and the

current wide usage of the X.25 recommendation in industry protocol.

Specifications for the X.25 Network Layer come from CCITT X.25 recommendations contained within the Yellow Book (1980 revisions), volume VIII. The X.25 subnet consists of the X.21 or X.21(bis) recommendation for the physical protocol implementation and initial circuit establishment. The data link layer protocol consists of LAP B procedures or equivalently the balanced mode operation of HDLC. At the packet level, the X.25 recommendation specifies a minimal set of required functions and services (115:73):

1. Call set-up and clearing.
2. Data and interrupt.
3. Flow control and reset.
4. Restart procedures.
5. Diagnostic information.

Optional functions are given in the recommendation and may be implemented at the users discretion. These options may be summarized as (100):

1. Packet window size or flow control negotiation.
2. Packet data size.
3. Datagram service.
4. Fast Select Service.
5. Throughput class negotiation.
6. Closed user groups.
7. One-way logical channels.

Certification of an operational X.25 packet network can be made through the use of network protocol analyzers which meet CCITT criteria

for X.25 packet network operation.

#### Test and Validation Requirements

The previous sections contained the requirements for UNID II, DELNET, and the X.25 protocol. Not mentioned was the criteria used to measure how the final design fulfills the given requirements (29). Within this Thesis, the term "validation" will be used as a process demonstrating the degree to which the design fulfills stated requirements. The term "verification", often associated with validation, implies an entirely different process, the process to measure the degree a design doesn't fulfill requirements. Tests for software and software/hardware integration will be tests limited to validating the software and will not attempt verification.

Phister developed a comprehensive test plan to validate UNID I operations within DELNET (84: Chapter 3). Childress developed a similar test plan for UNID II and validated portions of the UNID II software and DELNET/UNID II interface. Table 2-2 shows the test phases developed by Phister and Childress.



Table 2-2: UNID I and UNID II Validation Tests

	Phase	Phister for UNID I
x	I	UNID Local Side Data Transfer
x	II	UNID Network Side Data Transfer
x	III	Local-to-Network and Network-to-Local Data Transfer
x	IV	Local-to-Local Loop Back Data Transfer
x	V	Local-to-Local Data Transfer
x	VI	Network-to-Network Loop Back Data Transfer
	VII	Host-to-Host Data Transfer via 2 UNID Network
	VIII	Host-to-Host Data Transfer via 18 pair Cables
	IX	Host-to-Host Data Transfer via 3 UNID Network
		Childress for UNID II
x	I	Validate Inter-module Interface
x	II	Validate Module Integration
x	III	Validate Host-to-UNID II Data Transfer
	IV	Validate UNID II-to-LSI-II Data Transfer
	V	Validate UNID II-to-UNID II Data Transfer
		x - indicates the phase was fully completed

The testing in both cases was incomplete due to unavailability of hardware for an additional UNID. From both their test plans, useful guidance for test procedures validating UNID II may be extracted. The following paragraph presents the most distinct requirements necessary for software validation.

The stages of software validation should be partitioned by each of the layers of protocol used within the UNID II: the physical layer, the

data link layer, the network layer, and the internet layer. Each layer should have as a phase of validation: individual module interface validation, software configuration testing, system configuration testing, and system configuration testing. In the chapters that follow, Chapter III presents the system or protocol design of UNID II, Chapter IV presents the hardware and software implementations of the design, and Chapter V presents the validation of software and hardware/software integration.

The following section concerns the implementation the UNID II and DELNET within the structure of the ISO Reference Model protocol.

#### Protocols

Use of the ISO OSI Reference Model (21, 112) was proposed (33) for the overall DELNET protocol design. Since that time, protocols for DELNET are made by specifying their functions within the ISO Reference Model. With the Reference Model in mind, Phister developed specific protocols for DELNET (93) using the following standards and recommendations:

1. CCITT X.25, LAPB, in the data link layer.
2. CCITT X.25, datagram service in the network layer.
3. CCITT X.121, internet description in the transport layer.
4. TCP/IP in the transport layer for datagram service.
5. Federal Information Processing Standards (FIPS) and National Bureau of Standards in the transport layer for virtual service.

These specifications and recommendations make specific references

to the following associated specifications and recommendations(103:298-301):

1. X.1, specifies the class of service for users in public data networks which offer with respect to operating mode, transmission speed, and permissible character code.
2. X.21, Draws on X.1, X.24, X.26, and X.27 to specify the electrical and mechanical characteristics of a DTE/DCE interface.
3. X.24, specifies the logical definition of ten interface circuits used across analog communications circuits.
4. X.26, specifies the electrical characteristics of the DTE/DCE interface.
5. X.27, specifies the electrical characteristics of the DTE/DCE interface for speeds up to 10 mbps.

#### Implementation Issues in the ISO Model

Manufacturers of computer network interface equipment often set standards unique to their own product line and often overlooked interoperability between different the equipment of different manufacturers. An example of this situation is IBM's System Network Architecture (SNA) and Digital Equipment Corporation's DECNET. In reaction to these many incompatibilities, the International States Organization developed a communications nodal reference model, commonly referred to as the ISO Reference Model or ISO Model. This model clearly defines a standard international computer communications interface. The development of a common reference model was in part to promote a high degree of intercompatibility between different networks, to simplify

interface technical details, and to maintain fair business relationships between large and small equipment suppliers. Establishment of a model for interfaces within networks was a major step forward in developing an acceptable world-wide industry standard for network architecture. At this time the ISO Reference Model was established, several networks were already in wide usage. Among them were ARPANET, IBM's System Network Architecture (SNA), and Digital Equipment Corporation's DECNET. While the shaping of these particular networks do not fall precisely within the framework of the Reference Model, but they do provide a "close approximation" to the functional layering of the model. Figure 2-3 shows an approximation of how these existing networks fit into the ISO Reference Model.

Layer	ISO	ARPANET	SNA	DECNET
7	Application	User	End user	Application
6	Presentation	Telnet, FTP	NAU services	
5	Session	(None)	Data flow control	(None)
4	Transport	Host-host	Transmission control	Network services
		Source to destination IMP		
3	Network	IMP-IMP	Path control	Transport
2	Data link		Data link control	Data link control
1	Physical	Physical	Physical	Physical

Figure 2-3. Approximate Correspondence Between the Various Networks (112:22).

Since the advent of the ISO Reference Model, new area networks adhere, at least in principle, adhere to the seven layer model. Problems do remain, specifically with the adoption of the CCITT R.21 and R.25 recommendations for the subnet layers which were developed before

general use of the reference. Some manufacturers approached to this blurring of the subnet layers by marketing all-in-one packages which implement the physical and data link layers as a single product. Examples of this are Intel's Ethernet and STARLAN and IBM's PC Network (46).

One of the basic concepts behind standardized components and systems architecture is to increase compatibility between equipment from different manufacturers. Intel, IBM, and MicroSoft have developed the OpenNet trademark as adjoint cooperation in standardized components. Use of this trademark is restricted to equipment conforming to the ISO Reference Model. Presently OpenNet products exist for seven layers of the Reference Model. Common implementation of the physical and data link layers use the 82586 or 82568 single chip LAN in conjunction with the 80186 microprocessor to create a LAN essentially in two very large scale integrated (VLSI) chips. The complexity of networks using these LAN chips varies greatly, but in general conform to carrier sense multiple access/carrier detect (CSMA/CD) protocol using IEEE 802.3 standards. Ethernet and STARLAN are both examples of CSMA/CD network technology.

Above the subnet come software implementations of the transport and higher layers. The transport layer is where UNID II will interconnect with the world. OpenNet products currently make use of Intel's INA 960 transport layer software (68). The package actually implements both network and transport layers, but for the present, the network layer functions only as a null layer at present. Routing of message traffic goes directly from the data link interface to the transport layer software. Future releases of INA 960 (68) will include a functioning

network layer for multiple users. Figure 2-8 shows the conceptual details for INA 960:

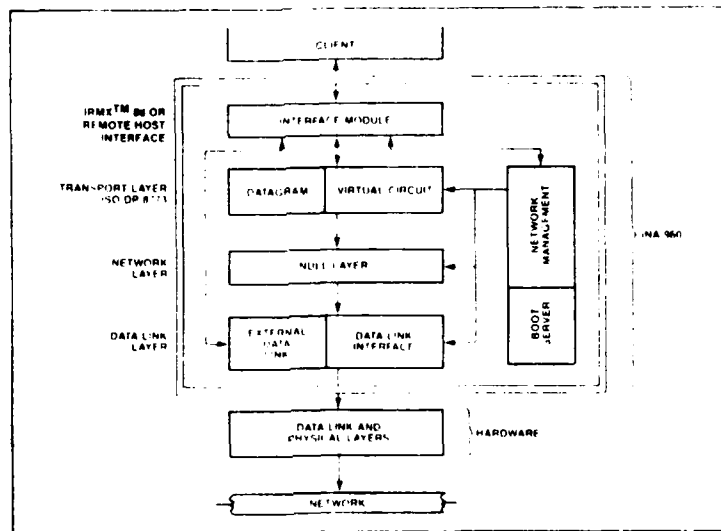


Figure 2-4. INA 960 Conceptual Diagram (68:27)

As shown above, the INA 960 has datagram, virtual circuit, and direct access to the data link layer. These access implementations compose horizontal slices of the transport layer. Intel's INA 960 is intended for general usage with non-Intel products. A tailored version of INA 960, INA 961 is designed specifically for Intel's own product line and meets the newly released ISO 8073 transport layer standard (46).

OpenNet normally constructs the session, presentation, and application layers located above the transport layer as a single product. The MicroSoft's XENIX operating system performs the operating system functions, and with the XENIX-NET software also performs transparent network access functions. Transparent network access allows the user to treat all interactions on the network as an access to a file. The

XENIX-NET software memory maps the file location in the network with the file name. XENIX-NET network functions include remote file access, network management, and communications with other operating systems.

Another OpenNet operation systems is Intel's RMX (68). The RMX operating system serves to:

1. Monitor system peripherals.
2. Control I/O with system peripheral devices.
3. Provides a user configured environment using multiprocessors.
4. Support common programming languages and software tools.

RMX is specifically designed for Intel's 8086, 8088, 80186, 80286 microprocessor product line. The OpenNet products for XENIX and RMX (XENIX-NET and RMX-NET respectively) provide interoperability between XENIX and RMX.

MicroSoft's MSDOX has also come into use within OpenNet through MSNET software (46). MSNET software allows personal computers running under MSDOS or PC DOC to run as an applications package to either XENIX or RMX operating systems. Thus, OpenNet products support the full seven layers of the ISO Reference Model and implement communications between different nodes and different networks using the XENIX, RMX, MSDOC, and PC DOS operating systems.

#### Conclusion

This chapter introduced the requirements for both UNID II and DELNET. Data flow diagrams from the SADT design of UNID I were referenced for UNID II requirements. DELNET requirements were summarized in Table 2-1. Then X.25 network recommendations were

presented as the ISO reference model protocol requirements for UNID II. Test requirements for validating software and hardware integration were then briefly described through the association of design implementation with the functional specifications of the ISO Reference Model. The next chapter, Chapter III, discusses both early UNID II hardware design and the current UNID II hardware implementation in the multibus system using the SBC 544 local module and SBC 88/45 network module. The chapter concludes with the discussion on the UNID II software as developed by this Thesis.



## CHAPTER III.

### System Design

#### Introduction:

This chapter presents the protocol design structure used by UNID II. The protocol is consistent with the design requirements given in chapter II. Specifically the design is developed from X.25, X.21(bis), and V-24 recommendations and incorporates physical equipment operating under EIA RS-232-C, EIA RS-422, and EIA RS-449 standards as well as the general constraints imposed by the ISO Reference Model. The first section in this chapter gives a brief synopsis of the UNID II and DELNET design by detailing the method UNID II will be incorporated into DELNET. The following sections elaborate on the UNID II protocol implementation by discussing each of the layers of the ISO Reference Model. The application layer, presentation layer, and session layer are discussed only briefly as they have no rigid definition by CCITT and are not actually incorporated into the UNID. The discussion continues with the transport layer, the layer that will communicate with the UNID. Next the Internet protocol is discussed. The Internet protocol is implemented to facilitate communications between the transport layer and the network layer. The subnet layers follow the Internet protocol discussion and represent the basic building blocks of the UNID II protocol. Their order of presentation is the network layer, the data link layer, and the physical layer. The network layer provides for the routing capability of the UNID. The data link layer controls the transfer of bits across the communications link. The chapter concludes

with a discussion of the physical layer implementation within UNID II.

### Synopsis

As with DELNET, UNID II will support the ISO Reference Model. Incorporation of UNID II as a node within DELNET will occur at the transport layer. The UNID will implement the network, data link, and physical layers. Hosts for UNID II require transport, session, presentation, and application layers. The host and UNID II layered software will be linked by an intermediate layer between the network and transport layer known as the internet layer. The host and UNID will implement the internet protocol as described by CCITT X.121 internet specifications for the TCP/IP transport layer datagram service.

In applying the CCITT recommendations, X.1 specifies the user class of service and X.2 specifies the user services and facilities required in public data networks. X.25 specifies the DTE/DCE interface between packet mode public data networks. In specifying the DTE/DCE interface, X.25 references X.21(bis) for non-digital circuit operation at the data link layer. X.21(bis) in turn references V.24 (115:44) for the functional characteristics of the physical layer. V.24 references the V.28 (115:44) for the electrical characteristics of a 25-pin interface and X.26 for the operation of a 37-pin interface. Within the US, 25-pin interface protocol is handled by the EIA RS-232-C interface and the 37-pin interface protocol is handled by the EIA RS-449/442 standards.

A summation of the ISO Reference Model as it applies to DELNET and UNID follows. The material, in part, comes from Childress (15:Chapter 2) and Phister (93:Chapter 2), and Tannenbaum (112).

### ISO Reference Model

The ISO Reference Model provides the framework for developing communications protocols for end to end communications between open ended systems. Each of the seven layers within the model group relates activities within a common layer. The model then assigns the layer of activities a hierarchy within the system to form a network. Each layer partitions an activity into easily defined tasks. The layers are stacked on top on one another to form a node within a network. The most fundamental tasks appear at the bottom while the most specialized tasks required of the user appear at the top. Organizations such as CCITT and the National Bureau of Standards have tailored their specifications and standards to be compatible with the OSI model (38, 39, 40, 41, 42, 43, 44). However, some existing standards, such as the CCITT X.21 and CCITT X.25 recommendation were established before general usage of the OSI model and do not "conform very well" to criteria established by the model. In fact, as shown by (3), these two recommendations must be divided into vertical, horizontal, and temporal sublayers to adequately fall within the OSI model.

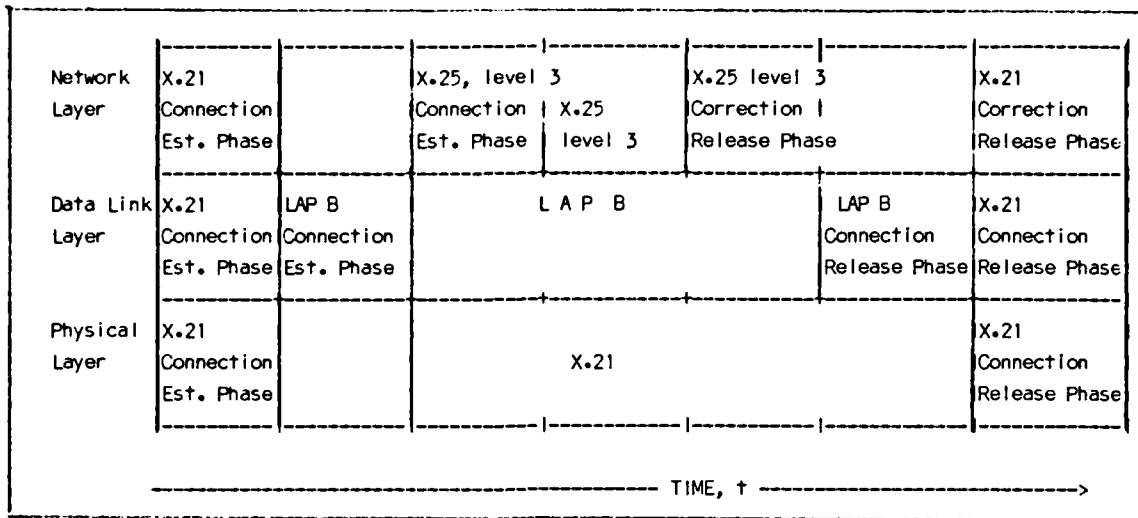


Figure 3-1. Temporal Order Sublayering in CCITT X.25 Recommendation (3:22).

Figure 3-2, below shows the seven layers of the OSI model as it would be implemented using four UNIDs to form a dual ring.

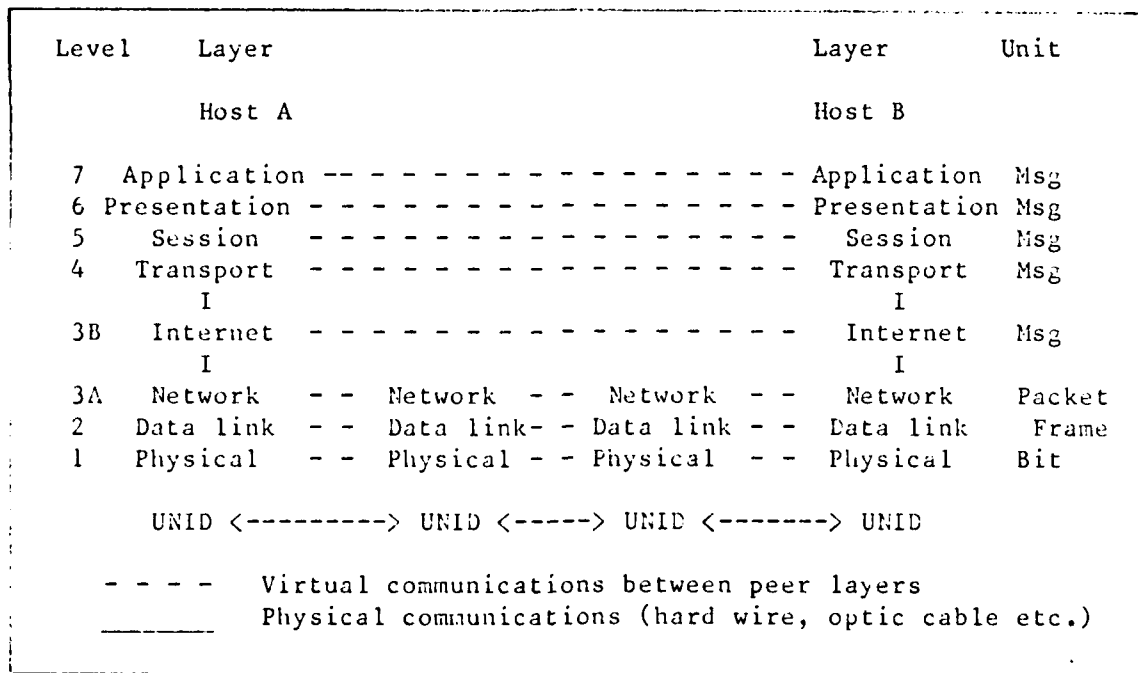


Figure 3-2. ISO OSI Reference Model Implemented With UNID (15:2-7).

Actual communications takes place horizontally (between hosts) at the physical layers (solid lines). Remaining layers form the same equivalent functions between nodes and communicate horizontally via a peer process. The dotted lines within the figure represent this peer process as virtual communications between layers. Messages travelling through a node must first travel vertically through different layers, each performing its specific task. Once at the physical layer, communications proceeds horizontally (DTE to DCE or DCE to DTE) to the next node. At the destination node, the message again proceeds virtually with each layer performing the reciprocal function of its associated peer process. In the specific case demonstrated in figure 3-2, Host A is separated from Host B by four different UNIDs within the DELNET. Host A and Host B may be on the same DELNET ring or on two separate rings in which case the inner two UNIDs would both have country codes of '0'.

The application layer process specifies user requirements, and to a large extent consists of a multitude of horizontal sublayer partitions within the application layer. Horizontal sublayers may consist of several user options such as file transfer, interactive communications, or electronic mail transfer. Once both nodes (near and distant ends) have agreed to jointly support the designated semantics, a single horizontal partition within the applications layer is selected and communications proceeds to the presentation layer.

The primary concern of the presentation layer is with the representation of the information given by or to the applications layer. Thus, while the applications layer concerns its self with the semantics

or meaning of the message, the presentation layer defines the syntax of the message (36:1403). The syntax may vary from graphical representations to various international character sets or any one of a various number of bit representations of semantics.

The session layer, and all layers below, are concerned primarily with the movement of the message and not its representation. Specifically the session layer performs the top most addressing functions within the host and the first structuring of the message syntax or Session Service Data Units (SSDUs) which segment the data (26:1397). Within the SSDU, Session Protocol Data UNITS (SPDUs) carry data or control information to and from the upper layers of the model. Below the session layer comes the transport layer which is the first layer having a clearly defined structure and format for operation within DELNET and UNID.

#### Transport Layer

Portions of the transport layer information presented comes from the previous work of Phister (93), Childress (15), and Hunt (37).

The transport layer is the only layer within the ISO model controlling the transfer of data between open systems. The transport layer structures the message from acceptance by the lower three layers (subnet) for eventual transmission across the network. At the virtual communications level within the model communications with the upper three layers proceeds at a specified or at least known quality of service. Functions provided by the transport layer included flow control procedures, ordering of message packets, and error checking. The operation of the transport layer may be described in three phases: connec-

tion establishment; data transfer; and connection release.

During the first phase, connections establishment, the transport layer provides a means for a pair of hosts to locate one another through the many possible physical connection schemes between them by the use of a user identity code. The completion of this phase establishes data transfer between users.

The second phase, the data transfer phase exchanges Transport Service Data Units (TSDUs), which as with the session layer may contain Transport Protocol Data Units (TPDUs) (37:187). The TPDUs establish in the connect phase the following classes of service: sequence, blocking, concatenation, segmentation, multiplexing or splitting, flow control, error detection and/or correction, and error recovery. Finally the data transfer phase must allow expedited packets (high priority packets) to move ahead of other packets in the system.

The third phase, the release phase, simply terminates the transport layer connection in an orderly fashion. Functions provided in this phase consist of notification of reason for release; identification of the transport connection released; and additionally requested information, as may be specified by existing protocol.

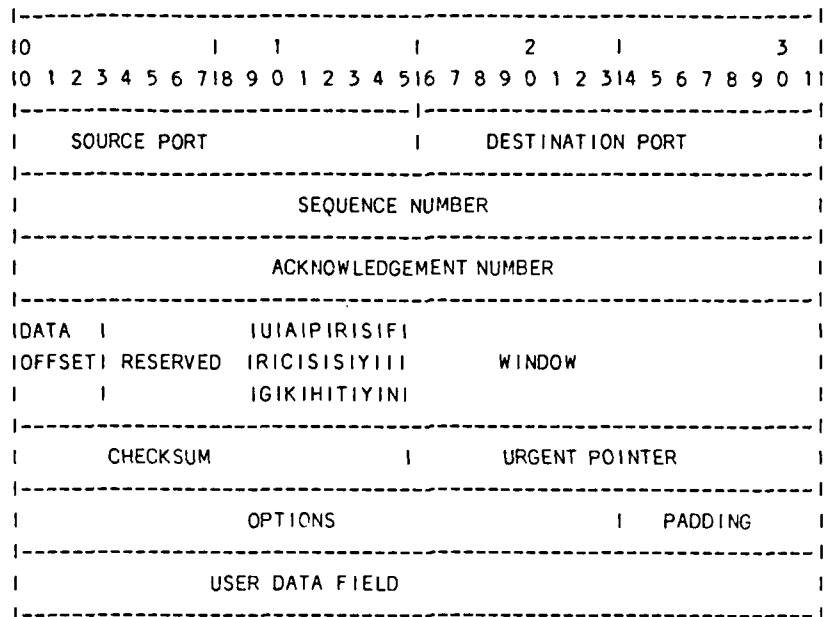
The transport layer as applied to DELNET follows both datagram services and virtual circuit services. For datagram service, the TCP protocol provides reliable service between host processors and the network nodes. The TCP interfaces between the applications layers and the subnet layers. Within DELNET, TCP will make up the lowest layer of the host and interface with the protocol structure embedded in UNID. The transport layer in DELNET will provide the Internet Header Format

(IHF) which forms the datagram header. The IHF resides in the host and not UNID so as to allow UNID the option of both datagram and virtual circuit service.

The virtual circuit operation for DELNET is defined in FIPS (38, 39, 40, 41, 42, 43, 44). As stated by Phister (93:2-17) "The reasons for this selection are: (1) it can be obtained to run on the VAX from the National Bureau of Standards (NBS) and (2) if AFIT can fully implement TCP in UNID, the NBS will certify it."

Shown below in Figure 3-3 is the transport header alone. The transport header consists of 24 bytes and is placed in the datagram data section following the IP header bytes.





- URG: Indicates use of URGENT POINTER field
- ACK: Indicates use of ACKNOWLEDGEMENT field
- PSH: Push function
- RST: Used to reset the connection
- SYN: Used to synchronize sequence numbers
- FIN: Indicates no more data from sender

Figure 3-3. Transport Header used with DELNET (78:C-20 - C-25)

As of this time DELNET does not operate in the virtual circuit mode due to the incomplete implementation of the subnet layers within UNID. The TCP implementation within DELNET is more completely described in (93: Appendix C).

The following section describes the Internet layer through physical layers as applied to DELNET and UNID. This material updates the previous work of Childress (15:2-8 - 2-15).

Internet Protocol

"The IP protocol is usually implemented between the transport and

network layers to interface networks with different protocols and data entity formats through a common standard protocol (112: Chapter 8). The most common implementation of the IP is where two different networks are connected through a gateway. A gateway is a set of computer hardware and software programs through which two different networks can communicate. The most common implementation of a gateway is where the functions of the gateway are divided in half and implemented at nodes of the two different networks. The IP protocol is implemented as the common protocol between the gateways and the next higher and lower layers, the transport layer and the network layer, respectively. The IP protocol is effectively sandwiched between the transport and network layers, forming another layer that could be called the internet layer, layer three-and-one-half. The sandwiched layer representation is shown in Figure 3-4 as applied to the ISO Reference Model in the UNID and DELNET. Although the IP is used between two different networks, it may also be used between a host and the host's servicing packet switching node. The IP protocol is implemented in the UNID and the DELNET (93:Appendix C, 15:2-8 - 2-9).

Level	Layer		Layer	Unit
	Host A		Host B	
7	Application	-----	Application	Msg
6	Presentation	-----	Presentation	Msg
5	Session	-----	Session	Msg
4	Transport	-----	Transport	Msg
3B	Internet	-----	Internet	Data-
	I		I	gram
	I		I	
3B	Internet	-----	Internet	Datagram
3A	Network	-- Network	Network	Packet
2	Data Link	-- Data Link	Data Link	Frame
1	Physical	---- Physical	Physical	Bit
	UNID	-----	UNID	-----

Figure 3-4. ISO OSI Reference Model with the Internet Protocol (IP). (15:2-9)

The figure below shows the IP header format. The IP header is 32 bytes long as is the transport header and starts in the datagram data section as the first byte of data. Hence, the maximum of 128 bytes of data allowed for a datagram has 56 bytes of transport header and IP header information leaving at most 72 bytes of data for user messages. Implementation of ISO layers above the transport layer also require additional bytes for their respective header formats.

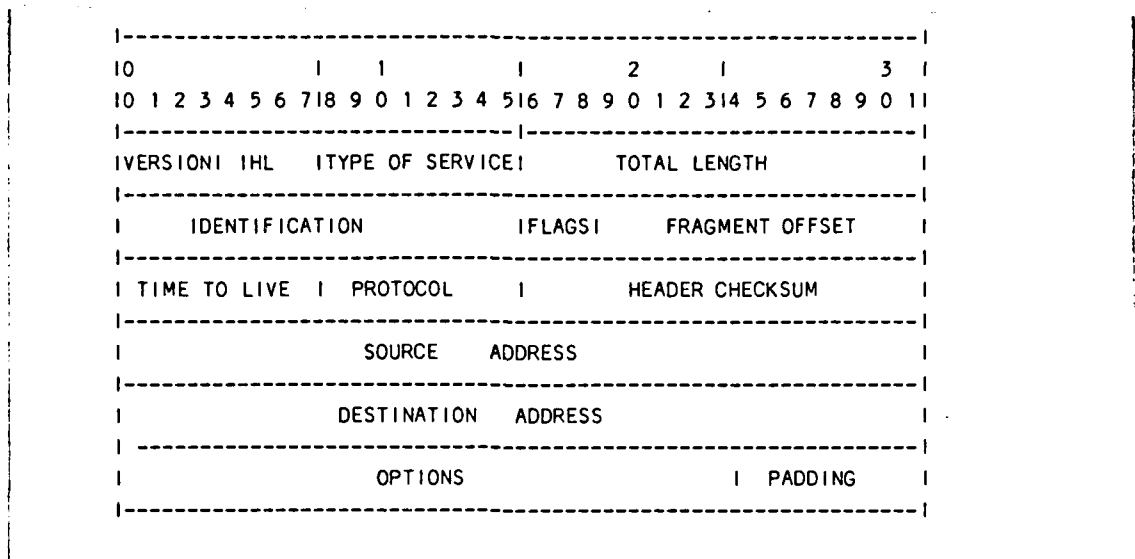


Figure 3-5. Internet Protocol using in DELNET (78:C-20 - C-25)

Subnet

The physical, data link, and network layers form the lowest three layers known collectively as the subnet. Subnet functions route messages to and from specified hosts. Once above this level, message traffic moves between a single host pair. Within the subnet, messages from any of the nodes on the network may be present, awaiting final delivery to either a host at the existing node or another node in the LAN. The subnet protocol specified for UNID and DELNET is defined by the CCITT X.25 recommendation. The use of the X.25 recommendation came about through DELNET's requirement for high compatibility between the many diverse computer systems and networks currently in existence. The X.25 recommendation defines conditions of interface between packet switched networks and has come into general acceptance by the international community. The X.25 recommendation specifies the protocol of the network and data link layers of the subnet and refers to the

CCITT X.21 or X.21 (bis) recommendation for the physical layer and link access protocols.

#### Network Layer

The top layer of the subnet, the network layer, directs message traffic within the subnet towards the designated host. To perform this task, the network layer must provide routing control, sequencing, and flow control. In networks with provision for only one host, the network layer often does not exist in its full implementation. For the true X.25 network, the network layer must determine which of many possible paths a data packet will travel to reach its destination. Implementation of static routing tables affords a simple solution to the routing decision process within the network layer. More complex methods require the node to update dynamic routing tables in an effort to optimize the network data flow.

The 1980 addition of the CCITT X.25 recommendation allowed three types of services: virtual call, permanent virtual circuit, and datagram. Virtual circuit service requires call establishment and release during each session within the network and is the most complex service in the network. Permanent virtual circuit service dispenses with call establishment and release by maintaining a completed circuit at all times. Datagram service provides even simpler communications by never actually establishing a session between nodes. Each packet is its own complete session without requirements of a call request packet prior to establishment of the session. Within the virtual circuit session a feature known as "fast select" allows the call request packet to contain data for the designated host. In the years between 1980 version of the

X.25 protocol and the revised 1984 version of the X.25 protocol, only the fast select option was implemented by a national carrier Nippon Telephone and Telegraph, Japan (NTT) (105:41), hence, the X.25 (1984) recommendation as described in the CCITT "Red Book" dropped the datagram service altogether, yet retained the "fast select" facility. The current implementation of the X.25 recommendation within UNID II has only datagram service. However, full compliance with X.25 protocol is planned for the final evolutionary stages of UNID II.

Datagrams are packets containing all address information required to route the packet to the eventual host. Moreover, datagrams contain the standard default of 128 bytes allowed for data within a packet and form the basic element of communications between two hosts without the required overhead for establishment of a virtual circuit. While fast select service also contains data in the call establishment overhead of a circuit, it is roughly 33% less efficient due to requiring three packets to complete a single data packet, while a datagram on the other hand takes only two packets to complete a transaction (8:18). More information on datagram service may be found in (8, 93, 112).

In all services (datagram, virtual circuit, and permanent virtual circuit), the X.25 recommendation specifies a common flow control procedure. Flow control is used as a means in delaying with congestion in the network. As pointed out in (112), flow control messages may provide information on the amount of available buffer space, a busy condition in the node, or a simple acknowledgement of data. Flow control packets have not been implemented within UNID or DELNET at this time.

The X.25 recommendation specifies a priority for messages destined

for the transport layer. The network layer provides a prioritized queue where transport messages jump to the front of other message traffic. Should the queue become full, the network layer does not accept any more packets until space in the queues is again available. Once vacancies occur, flow control packets acknowledge packets received in the queue.

The following figure shows the network header specified for a datagram data packet.

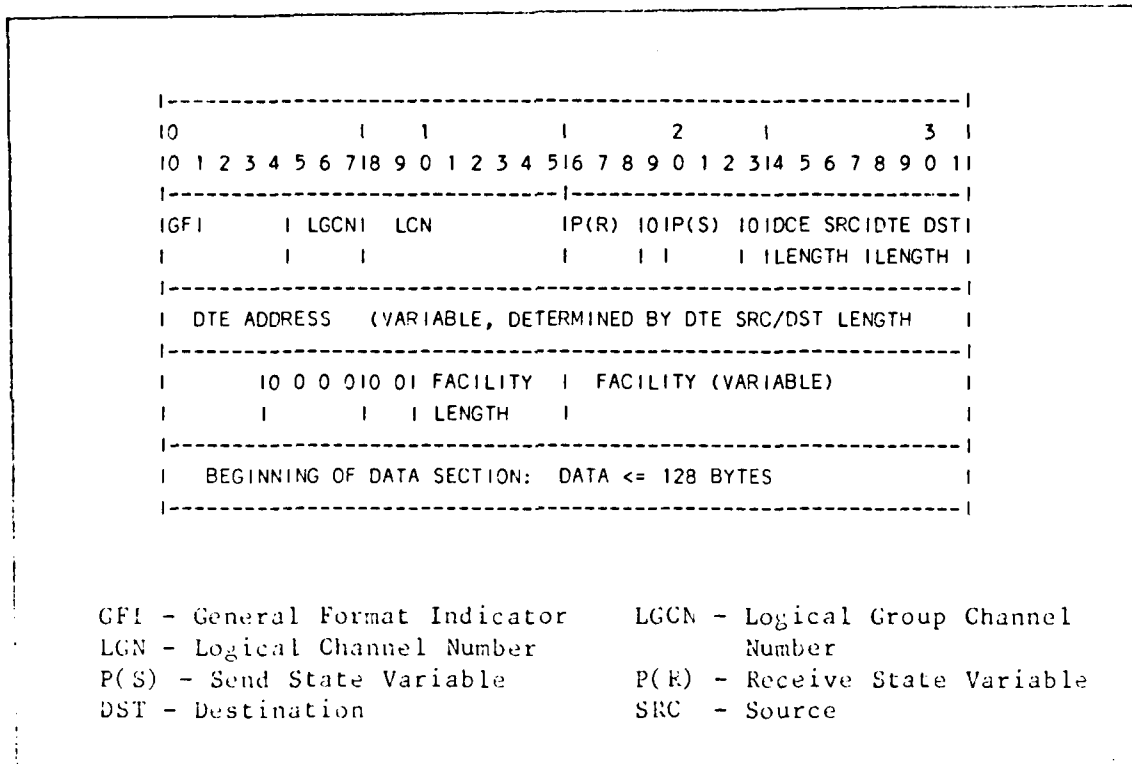


Figure 3-6. Network Header Defined by CCITT (115:100)

Data Link Layer

While the network layer deals with the exchange of messages in the form of packets, the data link layer's basic unit of exchange is the frame. The frame consists of control, and address information in addition to a complete packet. Services provided by the data link consist

of (18,112):

1. Initialization.
2. Identification of sender and receiver across the data link.
3. Synchronizing the decoding and encoding mechanisms.
4. Segmentation and delimiting the frames.
5. Data transparency.
6. Flow control.
7. Sequence and Error control through use of the cyclic redundancy check (CRC).
8. Abnormal recovery functions.
9. Data link termination.
10. Data link activation, deactivation and monitoring functions.

The data link uses a restricted set of the High Level Data Link Access Control (HDLC) called Link Access Procedure B (LAP B) as specified by the X.25 recommendation. LAP B allows only the balanced mode of operation contained in HDLC. Figure 3-7 shows the frame format specified by the X.25 standard. Figure 3-8 shows the X.121 format used by Hazelton, Phister, and Childress implemented inside the datagram header in lieu of the format specified by the OSI X.25 recommendation (15:2-13). The format will be changed to accommodate the X.25 format implemented in DELNET and UNID II.





software implemented for both UNID I and UNID II. The TCP/IP, developed for standardized use in the Department of Defense computer networks and a required protocol for DoD (21, 22). Computer networks, is partially implemented in the transport layer of UNID I and UNID II. The source and destination addresses are shown for each of the frame, packet, and datagram services where each structure is broken down into smaller segments to show its respective contents. The control (CT), country (CC), network (NC), host (HC), and port (PC) codes are shown for the implementation of the CCITT X.121 standard in the IP header used with the UNIDs. A more detailed description of the header structure and contents is in Appendix G (15:appendix D) and (93:Appendix C). Figure 3-9 shows the current frame format used for a more compatible implementation with the ISO X.25 recommendation.

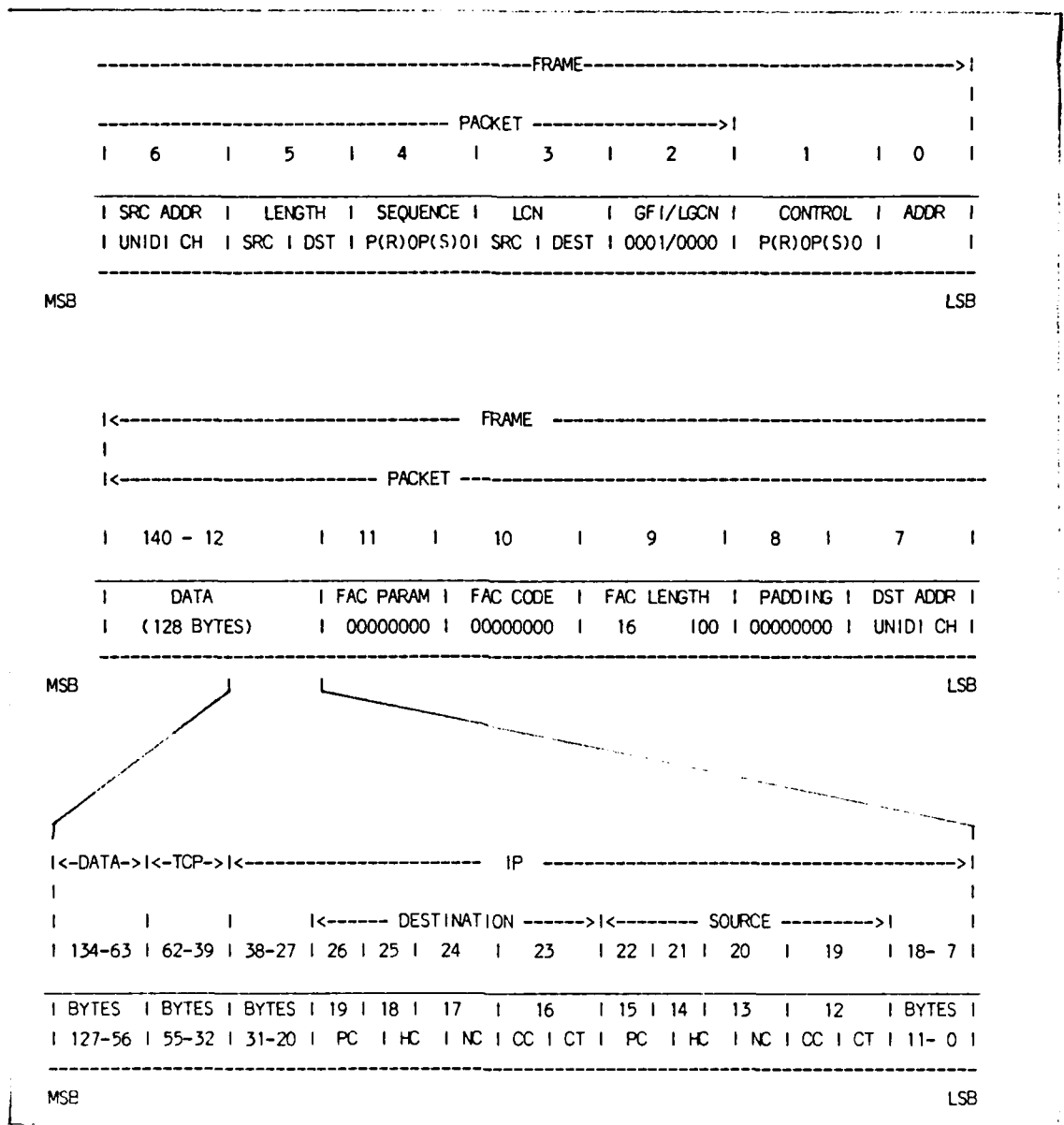


Figure 3-9. Currently Implemented Frame and Header Information

Since the data link layer can receive bad data from the physical layer, an error detection capability is included. A set of 16 bits based on cyclic redundancy check (CRC) calculations, called a checksum, is appended to the address in the frame as they are sent. The checksum is compared with the locally generated checksum at the receiving node.

When the checksums match, the received frame is assumed correct. A difference in the checksums indicates bit errors in the message. The CRC generator specified by the X.25 recommendation detects all single bit errors, double bit errors, odd number bit errors, and all 16 bit burst errors with some additional error detection capabilities (112:132). More details on CRC calculations may be found in (112, 80). Figure 2-3 does not show the flag or checksum bits, even though they are present, as these are automatically calculated, added, and deleted by the digital transmitter and receiver hardware at the network physical level. The flag bits are also appended to the data link layer frame. These flag bits are used for synchronizing the hardware to the beginning and ending of the data. Both the flag and the CRC bits are usually appended and deleted automatically by the physical level hardware. This hardware process is implemented in the UNID II hardware.

#### Physical Layer

The bottom-most layer of the subnet and of the ISO Reference Model is the Physical layer. It interfaces directly with the transmission median and is the layer most dependent on the type of transmission medium employed. The physical layer provides physical, electrical, mechanical, functional, and procedural services to define the physical interface between network nodes. These services include (81:1373):

1. Physical connections established at two or more data points.
2. Physical Service Data Units to preserve the data identity from one end of the link to the other.
3. Physical connector end points to terminate each end of the link.

4. Identification of data circuits which transmit and receive bits.
5. Sequencing the bits on the data circuit.
6. Fault condition notification by the physical layer.
7. Quality of service parameters.

The CCITT X.21 standard comes very close to establishing media independent protocol procedures. However, the standard is defined for true digital communications and has not come in wide usage due to the lack of truly digital systems. The X.21(bis) recommendation on the other hand was created for digital equipment interfaced by a synchronous modem communications equipment. Electronic Industries Standard (EIA) RS-232-C and RS-449 contain similar procedures. EIA RS-232-C specifies 21 interchange circuits, their electrical characteristics and use. EIA RS-449 was designed to replace the EIA RS-232-C interface by providing improved performance with longer interface cables, higher data rates, additional functions, and a tighter specification of the electrical standards. RS-449 specifies the protocol for 39 interchange circuits. The actual electrical characteristics are defined in RS-422 for balanced communications and RS-423 for unbalanced operation. An important characteristic of RS-449 is its compatibility with the more widespread RS-232-C interface (7). Within the United States, the EIA RS-232-C is the most often used of the standards and was chosen as the interface for UNID II's local module. The EIA RS-449 standard offers significantly higher data rates (up to 10 mega bits per second) and was selected for this reason for UNID II's network module. Original UNID II development (101) included a 20 milliamper (ma) current loop (part of the original RS-232a standard). This feature was not retained for UNID II operation as the

RS-232-C standard establishes slightly different electrical connections. Most of the older equipment employing the 20 ma current loop have been replaced with newer RS-232-C specification. This has also occurred in the military communications environment with adoption of MIL-STD-188. Appendix B shows the actual RS-232-C and RS-422 signals to be used in DELNET and UNID II (15:Appendix B).

#### Conclusion

This chapter began with a synopsis describing how UNID II's protocol falls within the ISO Reference Model. The sections that followed described each of the layers of the ISO Reference Model as they would be implemented within the UNID environment. The application layer, presentation layer, and session layer were discussed only briefly as their implementation is independent of the UNID construction. The next section in the chapter discussed how the transport layer operated using Transmission Control Protocol. The section following the transport layer described the Internet Protocol's interconnection between the transport layer and the network layer. The last sections in the chapter cover the subnet layers, namely the network layer, the data link layer, and the physical layer. The network layer implements the routing capability within the UNID while the data link layer implements the control necessary for transfer of frames across the communications networks. The final section in the chapter covered the physical layer and detailed the electrical characteristics of the interface used to interconnect the network nodes. The following chapter, Chapter IV continues with a detailed look at UNID II hardware design software implementations.

CHAPTER IV  
HARDWARE DESIGN

Introduction

This chapter presents the early UNID II hardware designs developed by Gravin (31), Palmer (91), and Matheson (84) and the present hardware design developed by (15). The first section describes the early hardware development. The next section details the present hardware design developed by Childress (15:Chapter 4). No changes in the hardware configuration have been made to Childress' design as the objective of this Thesis is to broaden the base of the software developed for UNID II. Three separate sections present the detailed composition of the SBC 544, SBC 88/45, and the SBC 86/12A. The last section of the chapter discusses how the RS-422 interface connects to other UNID IIs.

Initial Hardware Design

Gravin's Thesis in 1981 studied developing a UNID design with performance characteristics superior to the existing UNID I design based upon eight bit Zilog Z80 architecture. Based on Intel 8086 16-bit architecture, Gavin's design became known as UNID II. The objectives within the design were to improve delay and capacity characteristics of UNID through the use of 16-bit components and to increase design flexibility through the use of relocatable code. The preliminary design Gravin arrived at had two subsystems, a network subsystems and a local subsystem, operating from a common system bus known as a multibus. The 8086 and 8089 processors along with a private bus, memory, and I/O hardware comprised the architecture of the network module. A separate

8086 with four channels of serial I/O made up the local module. Gravin's design placed a strong emphasis on maintaining a high degree of functional and physical separation between local and network subsystems. The network subsystem's private bus allowed handling network I/O separate from activity on the system bus. The local subsystem then could make use of the system bus to handle local I/O activities. Interconnections between the local and network subsystems occurred through restricted system bus activity and a block of common memory. The common memory facilitated processor-to-processor data transfers through the use of first-in-first-out (FIFO) buffers separate from the transmit and receive buffers. Control of data transfer and synchronization within the the common memory occurred through the use of pointers and semaphores inside block memory headers. Gravin's functional analysis of UNID requirements led to development of data flow diagrams (DFDs) given in Appendix A. Though Gravin's original design has gone through many alterations, the DFD's provide accurate and useful guidance for all following UNID II development. Further information on the DFDs may be found in (31).

The hardware implementation of Gravin's design became three circuit cards within an Intel multibus card cage. Figure 4-1 shows the hardware construction.



An Intel SBC 86/12A processor was used as the local processor controlling the multibus to both the local and network subsystems. The remaining portion of the local subsystem and the network subsystem were constructed from wire wrapping components onto multibus cards. The parallel port off the SBC 86/12A was used to connect the four channel serial I/O card required for the local subsystem. The network subsystem, the 8086/8089 card, originated from an Intel applications note (72). Host-to-UNID data transfer would make use of one of the four serial channels on the local subsystem. The SBC 86/12A would perform packet construction and then place the messages into the common system memory. After the message was placed in common memory, an interrupt to the network subsystem would activate processing by the network subsystem. The 8086-8089 card would then perform the final frame construction on the message and send the message to the 8089 network I/O hardware.

Though 16-bit processors were manipulating the data, all data transfers were essentially eight bit data transfers. Data transfer between the SBC 86/12A and all hosts was through eight bit universal synchronous/asynchronous receiver/transmitters (USARTs) and the eight bit parallel port of the SBC 86/12A. Communications between the SBC 86/12A and 8086/8089 card was through 16-bit multiple protocol communications controllers (MPCCs). These 16-bit controllers, in fact, handled only eight bits of data with each data transfer. The remaining eight bits was used for status information (27:5-267). The end result was that all data transfers between local and network subsystems, the host-

to-UNID interface, and the UNID-to-UNID interfaces were all eight bit data transfers (15:3-4). A review of available MPCCs in late 1984 showed no commercially available MPCCs capable of 16 bit data transfers, other than a few specialized, but otherwise unsuitable integrated circuits. Nonetheless, the implementation produced a functional, though only minimally operational system.

Palmer's continuation of UNID II development was hampered by wiring errors in the network subsystem. This was discovered in the network board checkout phase (79:Chapter 4). At the conclusion of Palmer's design implementation, the only software developed for UNID II was for testing circuit boards and not for implementing any of the required communications protocol.

Matheson continued development of UNID II in 1983 (84). He determined that the original design would not function as specified. The main problems were in the 8086/8089 network board (84:3-11). The 8086/8089 network board could not handle two MPCCs as originally believed. Each MPCC required a separate interrupt to operate the transmit and receive buffers in the full duplex mode. Matheson's solution was to replace the 8086 on the network board with another 8089. With this design modification, the two 8089s acted as a cluster I/O processor with sufficient interrupt capability to handle both MPCCs. Another problem with the network board was an incompatible memory address mapping of the network card with the local card. This was corrected by placing memory and I/O ports of the network card in the system memory space. Matheson's final design is shown in figure 4-2 (84:1-9)

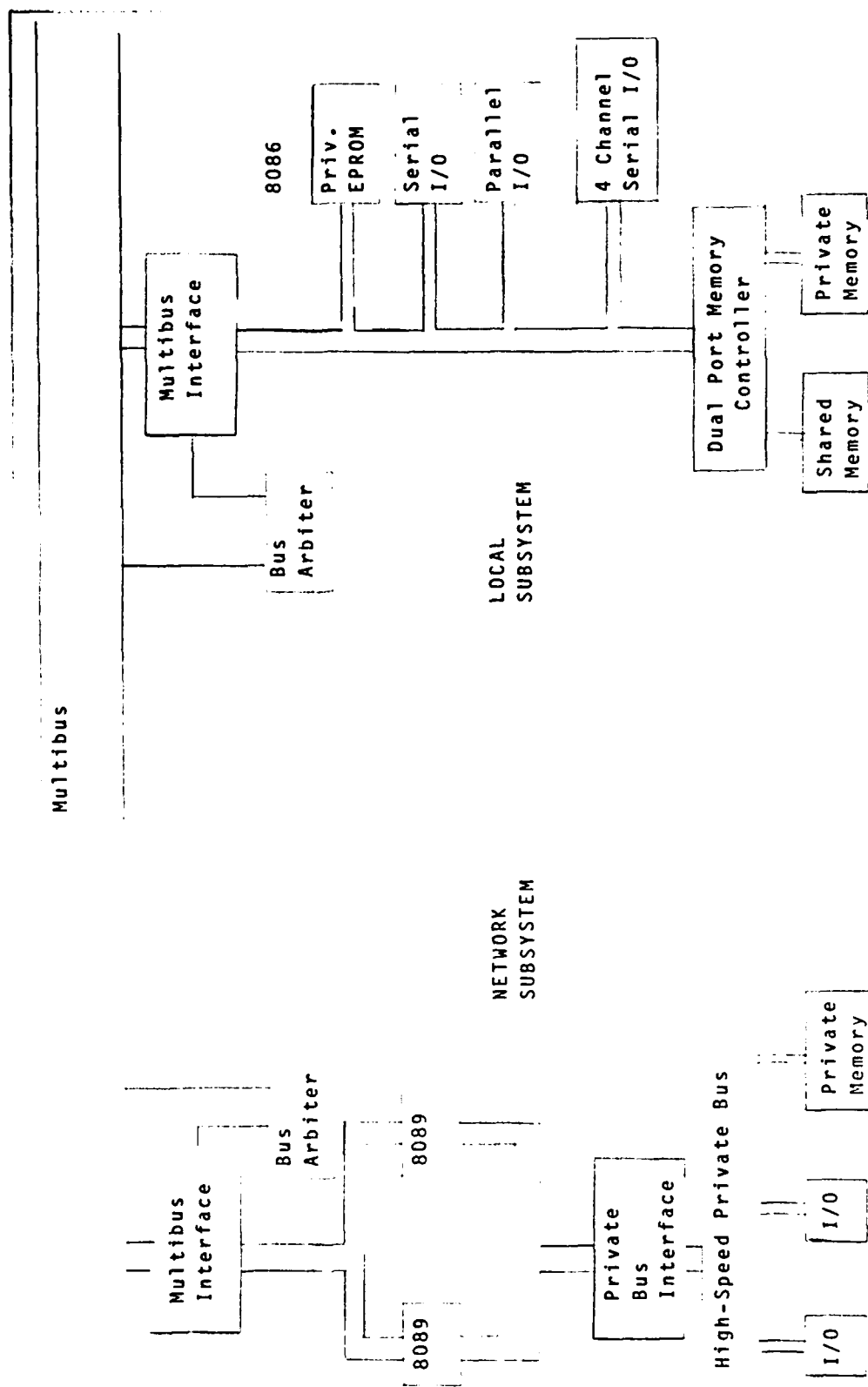


Figure 4-2. UNID II Block Diagram (Revised) (84:1-9)

After redesign of the network board, Matheson places a monitor program into the SBC 86/12A's EPROM to minimize the time consuming process of loading the system memory with the in-circuit emulator (ICE) 86 (43). He also translated UNID I software developed in PL/Z to PL/M and concluded his work with the functional validation of the local subsystem hardware and software. The UNID I code converted to PL/M, however, was only cursorily validated. It was left for (15) to validate all network and local software converted to PL/M.

#### Childress' Design

Childress' design used a architecture structure similar to Matheson's design, but took an entirely different approach in the implementation through the use of off-the-self SBCs. The design consisted of multibus card cage and power supply, an Intel SBC 544 board to implement the local subsystem functions, and an Intel SBC 88/45 board to implement the network subsystem functions. The SBC 544 board comes complete with four 8251A USART channels which may be configured as RS-232-C interface ports via software. The 88/45 board has two high speed serial ports which may be configured as RS-422/499 ports. The choice of Intel components is a matter of convince and availability and not an endorsement. Other manufacturers, such as Advanced Micro Devices (AMD) and Interphase, manufacture similar equipment. One particular board, not available at the time of Childress' selection for hardware design, completely implements a token ring passing node from the physical layer all the way to the transport layer (Interphase). As with UNID, most of the commercially available hardware makes use of the ISO X.25 recommendation for subnet protocol. Figure 4-3 shows Childress' UNID II architecture.

Note the design consists of only two modules, a local module and a network module, as indicated by the functional analysis performed by Gravin.

The SBC 544 local module consists of an 8085 processor servicing four interrupt driven 8251A USARTs. Up to 32K bytes of memory may reside on the SBC 544. Up to 16K bytes of this memory may be located in common with the multibus system memory. The board is implemented as a stand along processor requiring minimal assistance from the system bus. Resident within the SBC 544 module is the network layer software and network layer tables, pointers, and semaphores. More details for both the SBC 544 board software and SBC 88/45 software are given in Chapter V.

The SBC 88/45 network board consists of an 8088 processor servicing two high speed multiprotocol serial controllers (MPSCs) configured as two RS-422/449 ports. Direct memory access (DMA) operations between the central processing unit (cpu) and MPSCs allow data rates up to 800 K baud. A total of 64 K bytes of memory is available on board with up to 12 k bytes of memory shared with the system memory. Resident within the SBC 88/45 memory is the data link software and data link local tables. Each of the two SBCs provides the spare counter/timers necessary for the time-out clocks required by the X.25 protocol. Specific details of SBC 544 and SBC 88/45 operation may be found in (57, 58).

The use of the SBC 544 and SBC 88/45 allows simplification of the design procedure. The hardware test and debug phase of development may be assumed complete as all hardware is a commercially available piece of equipment. Moreover, software development aids operating on the Intel

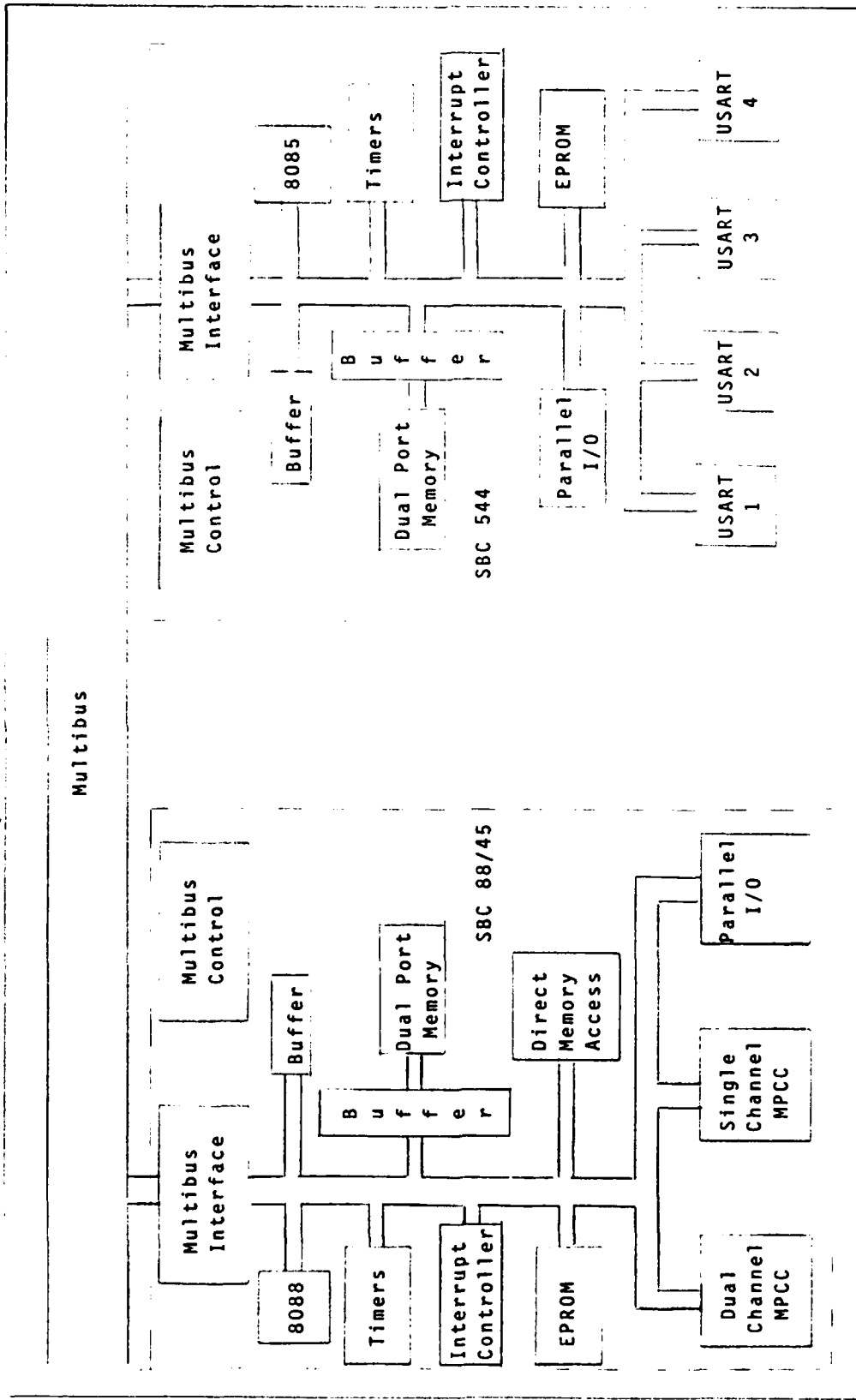


Figure 4-3. Current UNID II Block Diagram (15:3-7).

System 230 and System 310 are available. These software development aids include a full screen text editor; PL/M compilers, linkers, and assemblers; and program debugging aids to assist in program development. Detailed Hardware Descriptions of the SBC 544 (58)

The SBC 544 communications controller (58) is designed as an intelligent slave to a system processor. The board provides four serial (RS-232-C) ports, one parallel port, seven programmable timers, eight programmable priority interrupts, and an on board 8085 processor. The four serial I/O channels provide programmable character length, sync character (for synchronous operation), baud rate, parity, and stop bits (for asynchronous operation) through the use of four 8251A USARTs. Two 8253 Programmable Interval Timer (PIT) chips provide six count down timers. Four of the PITs are used for the USARTs while the remaining two are available to generate interrupts or as auxiliary transmit clocks. The programmable interrupts are generated from an 8259 Programmable Interrupt Controller (PIC); a Flag byte in the base memory address of the SBC 88/45; carrier detect interrupts to the 8085 CPU; ring indicator interrupt, also to the CPU; multibus interrupts, and timer interrupts to the 8085 CPU. The SBU 544 supports up to 8 k bytes of ROM/EPROM. Currently, the 6 k bytes of program code resides in two 2732A EPROMs. The code location is from the base memory of the board with the dual port dynamic RAM located beginning at 8000 H. If required, the SBC 544 may access additional system memory for its own use. The SBC 544 has up to 16 k bytes of dynamic dual ported RAM. The dual ported RAM allows access by either the SBC 544 CPU or a multibus master processor. The dual ported RAM is mapped into system memory

beginning at location 10000 H. In addition to the 16 k bytes dynamic RAM, 256 bytes of static RAM are available on the 8255 PPI beginning at address 7F00 - 7FFF H. The start address of the dual ported RAM is fixed at 8000H.

Detailed Hardware Description of the SBC 88/45 Advanced Data Communications Processor Board (ADCP) (57)

The SBC 88/45 is based on the 8088-2 8 bit processor and supports Serial Data Link Control (SDLC), High-level Data Link Control (HDLC) with data transfer speeds up to 800,000 baud. The SBC features the following characteristics:

- 1) 8088-2 CPU operating at a clock rate of 8 Mhz
- 2) 16 k bytes of RAM with up to 12 k bytes allocated for dual port operations
- 3) Up to 64 k bytes of ROM/EPROM
- 4) Three serial interfaces capable of RS-232-C or RS-422 operation (only channels A and C are capable of RS-422)
- 5) DMA access for up to two serial channels
- 6) Asynchronous, bisynchronous, SDLC, HDLC protocols
- 7) 9 levels of interrupt detection
- 8) 6 programmable counter/timers (4 are dedicated to the serial I/O port)
- 9) Compatible with RS-232-C, RS-422 and CCITT V.24 interfaces
- 10) Operates as a bus master or intelligent slave

The SBC 88/45 has an internal bus for on board memory and I/O operations which allows for parallel processing when used as part of a multibus system. The dual port RAM may be located at any 16 k byte boundary addressed by the on board CPU. The three serial interfaces are



independently programmable through both an 8274 Programmable Multiple Protocol Serial Controller (MPSC) and an 8273 SDLC/HDLC protocol controller. Channels A and B of SBC 88/45 are implemented through the 8274 MPSC. Channels A and B can achieve data transfer rates of 800 k baud in synchronous operation and 19.2 k baud in asynchronous operation. Channel C can achieve data transfer rates of 64 k baud in synchronous operation. An 8237-5 DMA controller provides DMA operation for two channels of communication with the SBC 88/45 board. Of six available count down timers, four are used for programming baud rates while two remain available for user programs. An 8259A PIC provides eight levels of programmable interrupts. A non-maskable interrupt is provided for catastrophic error recovery as would be required by a power failure.

There are seven large scale integrated (LSI) devices that require programming on the SBC 88/45: an 8255A PPI, an 8273 protocol controller, an 8274 MPSC, an 8237-5 DMA controller, two 8254-2 PITs, and an 8259A PIC. The sequence of programming these devices may be found in (57:Chapter 3).

The 8273 protocol controller supports asynchronous and synchronous operations in point-to-point and loop configurations. Up to 16 different commands allow the 8273 protocol controller to perform DMA functions between itself and the DMA controller. For data transmission, a begin flag, two byte checksum, and ending flag are added to the out going frame. For incoming frames these same bytes are removed. Following a frame reception, the 8088-2 CPU must check the 8273 status register to determine the frame checksum results. The 8273 protocol controller will act as one of the two RS-422 high speed data channels.

The 8274 MPSC has two identical channels which require the programming of 11 parameter registers. The 8274 MPSC performs the basic serial-to-parallel and parallel-to-serial data transfers necessary. The 8274 may operate in bisynchronous, SDLC, or HDLC protocol. Channel A of the 8274 MPSC may operate as either a RS-232-C or RS-422 serial port, while channel B is restricted to RS-232-C operation. The UNID II design will use channel A as one of the two RS-422 high speed serial ports to the DELNET ring, while channel B will be used as a monitor port for software development and for network capacity monitoring. More information about channel B operation may be found in chapter V. The data link layer software developed in Chapter V defines the UNID II as having two high speed RS-422 channels, channel A and channel B. Future references to the physical channels A, B, and C will be through the data link nomenclature used in Chapter V. Channel A data link software corresponds to the channel A port of the 8274 MPSC. Channel B of the data link software corresponds to the channel C port and is the 8273 protocol controller. Note also that the software requires channel A be configured as a DCE device and channel B be configured as a DTE device.

The 8237-5 DMA controller has four channels to provide direct memory access for two of the three serial interfaces on the SBC 88/45. Two of the four channels of the 8237-5 are dedicated to channel A of the 8273, while the other two channels may be programmed for transmit and receive functions to either of the other two serial ports. UNID II will use those remaining two ports for the 8274 MPSC for the second high speed serial port. Two 8254-2 PITs are available on the SBC 88/45. Two of the three timers/counters available on each 8254-s PIT are

dedicated to the transmit and receive clocks of the 8273 protocol controller and 8274 MPSC. The remaining timer on each chip is available for a programmable interrupt.

The 8259 PIC controls the interrupt services to the 8088-2 CPU. Interval capabilities include on board I/O, expansion I/O, multibus interrupts, and the SBC 88/45 Flag byte.

#### SBC 86/12A Hardware Description (56)

The SBC 86/12A is currently used only as a monitor resource for the memory operations performed by the SBC 544 and SBC 88/45 boards. Current designs do not include it's use in a functional UNID II.

The SBC 86/12A has an 8086 16 bit on board CPU with an internal bus structure for parallel processing functions on a multibus system. Up to 32 k bytes of RAM may be located on the SBC 86/12A of which all or part may be accessed by an off board multibus master processor. Four receptacles are available for up to 16 k bytes of ROM/EPROM memory. Presently a monitor routine uses 8 k bytes of EPROM on the SBC 86/12A. One serial RS-232-C port and one parallel port with 24 programmable lines resides on the SBC 86/12A. Communications with the monitor routine takes place through the serial port and a H-19 terminal.

#### AM 95/6445 Card Cage Description

The AM 95/6445 card cage is a six slot multibus with a control panel, 180 watt switched power supply, and two fans for forced air cooling. The card cage allows priority assignment of boards residing in the card cage. Presently, the SBC 86/12A board is placed in slot J-1 with the highest priority. Slots J-2, J-4, and J-6 are not used. The next highest priority is assigned to slot J-3 were the SBC 88/45 is

placed. No priority assignment is given to slot J-5 where the SBC 544 normally resides. When the SBC 88/45 is not tested, the SBC 544 resides in slot J-3. Jumper positions are given for the AM 95/6445 in Appendix C.

UNID II Physical Layer Interface

The physical layer interface of the UNID II consists of the two RS-422 channels on the SBC 88/45 used to support two different UNID rings. In full abstraction, The UNID II has two separate rings passing messages. One ring passes messages in a clockwise direction, and the other ring passes messages in a counter clockwise direction. Two entirely different approaches are possible in the configuration of the RS-422 interfaces. One is to dedicate channel A for processing messages in one direction and dedicate channel B for processing messages in the other direction. This technique would have every channel A connected to another UNID's channel A. Figure 4-4 show graphically how this technique would work.

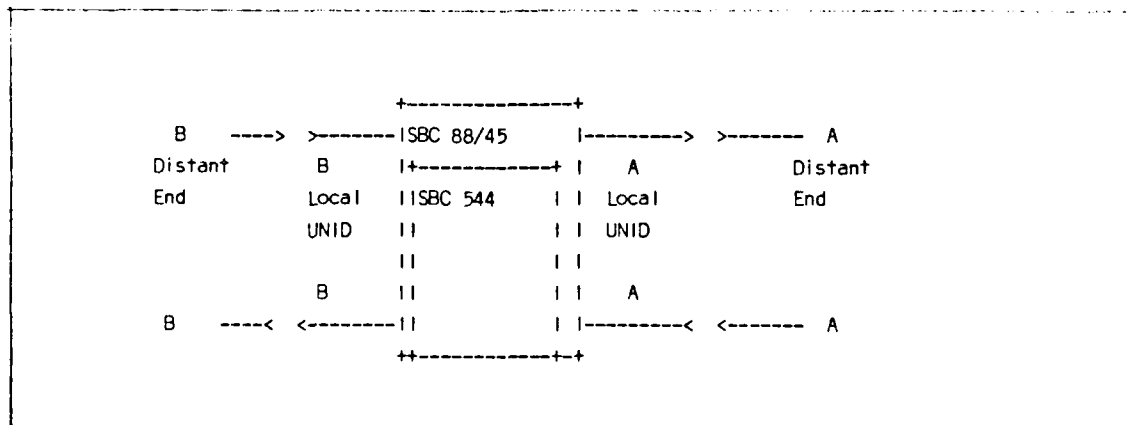


Figure 4-4. A RS-422 Inter-connection Technique for UNID II.

This technique has two limitations. One UNID would have to have a null modem interface built on both channels while the other UNID would have no null modem interfaces on either connector. A null modem simply interchanges the transmit and receive signals. However, since a RS-422 interface has balanced lines, most of the signals used would have to be interchanged. The result of this technique would be the creation of a UNID designed as a DCE Device and another UNID designed as a DTE device. Moreover, every DELNET ring would require an even number of UNIDs. The other limitation associated with the interface is that it would be impossible for hardware loop back on a single UNID. Such a capability would be useful for trouble shooting hardware problems. Another possibility would be to create a DCE interface and a DTE interface on the same UNID. Thus, a UNID could connect to itself in a hardware loop back or to any number of other UNIDS. This configuration would require one channel to have a null modem on each UNID II and each UNID II would have an identical hardware implementation. Figure 4-5 shows how this configuration would look.

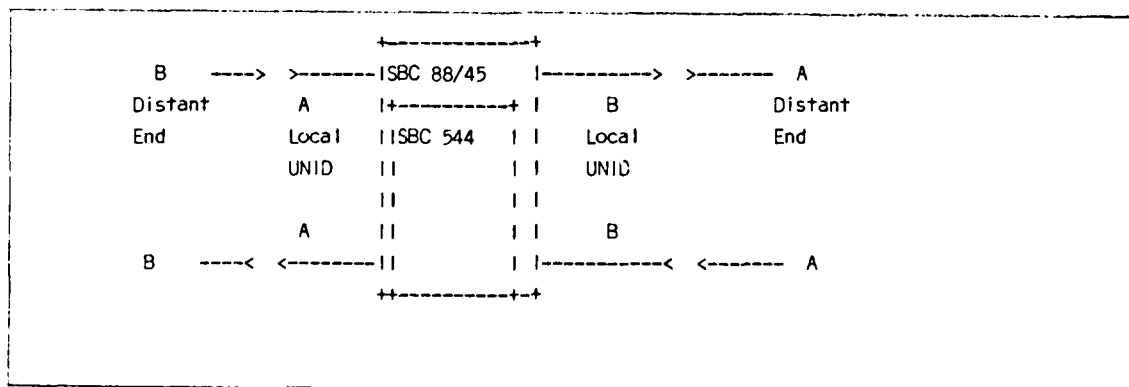


Figure 4-5. The Implemented RS-422 UNID Interconnection Technique

The present hardware and software configurations implement channel A as a DCE device and channel B as a DTE device. Refer to (Appendix B) for more information a null modem configuration.

#### Conclusion

This chapter presented the hardware design evaluation of UNID II. First, the designs of Gravin (31), Palmer (91), and Matheson (84) were briefly given. Then, the final hardware implementation developed by Childress (15) was developed in detail. The information discussed concentrated documentation of the current hardware implementation of the UNID II. The final section in the chapter discussed the hardware RS-422 interface as it is currently implemented on the UNID II. The following chapter, Chapter V, describes the UNID II functional testing procedures and the results obtained from those tests.

CHAPTER V  
SOFTWARE DESIGN

Introduction

This chapter briefly presents the design completed by the past Thesis effort (15) and then presents the design and implementation of the CCITT X.25 recommendation which followed. The contents of the chapter focuses primarily on the design and implementation of a protocol meeting the CCITT X.25 recommendation. This chapter is composed of the following sections: previous development, program language selection, data structures, UNID II network layer software development and implementations, data link software development and implementation, and physical layer implementations.

Previous Development

The PL/M programming language was used for development and implementation of the past UNID II software (15:4-1 - 4-2). The previous selection of PL/M was based primarily on two key criteria:

- 1) It was the only high level language available for both the 8085 and 8088 processors used in the UNID II hardware.
- 2) The developer (15) had prior experience with PL/M.

Development prior to Childress made use of several different languages. Among them were "C" (84) and PL/Z (93). The PL/Z software written by Phister (93) was translated to PL/M by Childress, then modified for operation in the current UNID II hardware design.

The PL/M code for the 8080 and 8085 processors (PLM80) did not have the facilities for dynamic memory allocation. Hence, indexed arrays

were used in lieu of pointer based linked-list data structures. Moreover, previous implementations of the UNID I software had also used indexed arrays. This commonality of UNID I and UNID II software allowed the previous software modules and basic program design to be integrated into UNID II's hardware. The overall effect was a savings in UNID II development time.

Communications between the SBC 544 and SBC 88/45 was addressed through the use of semaphores (Appendix E). The use of semaphores by each processor eliminated the problem of two separate processors trying to access the same memory location at the same time. This measure of protection assured each processor only valid data would be manipulated.

UNID I data flow between receive and transmit tables (memory buffers) of the network layer software was altered (15:Chapter 4) in the UNID II design. Local to Network tables (LCNTB), Local to Local tables (LCLCTB), Network to Local tables (NTLCTB), and Network to Network tables (NTNTTB) were eliminated in favor of direct memory transfers. The removal of these four tables from the network software simplified the software and decreased processing time for each message. Most of the development was limited to the network layer TCP/IP routing software. The data link simulation software did not contain flow control changes implemented in the network software. Additionally, the Local to Network and Network to Local tables remained in the data link simulation software. Finally, the data link software also did not include the semaphores developed for the network simulation and operational software (see figures 5-1, and 5-2).



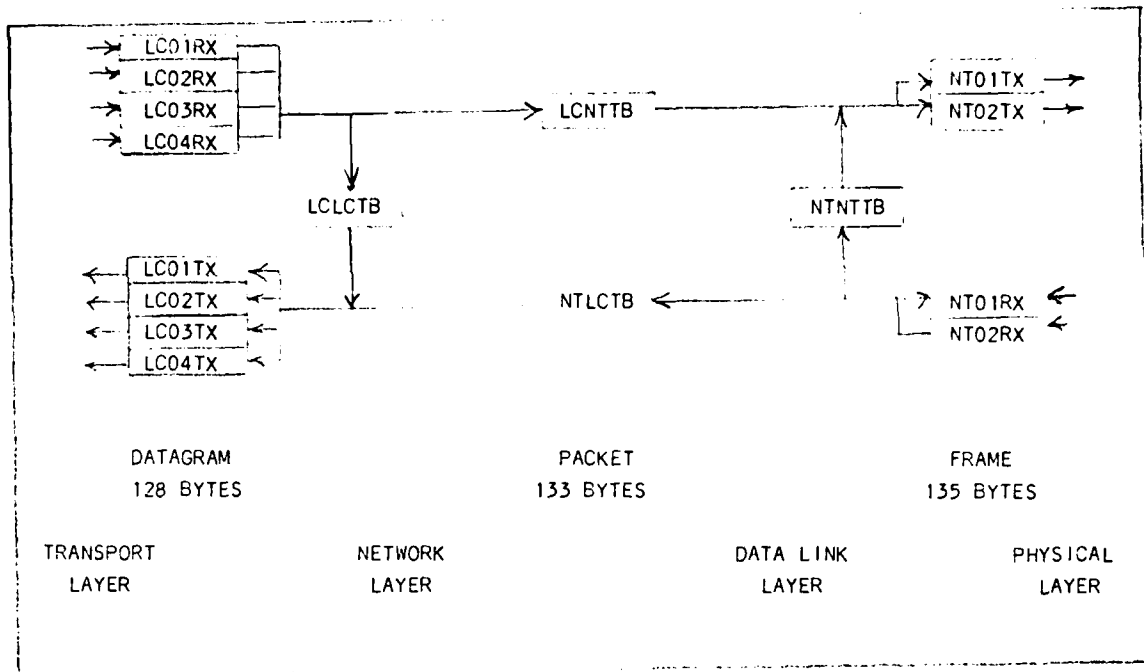


Figure 5-1. Original UNID Data Structures and Flow (15:4-4).

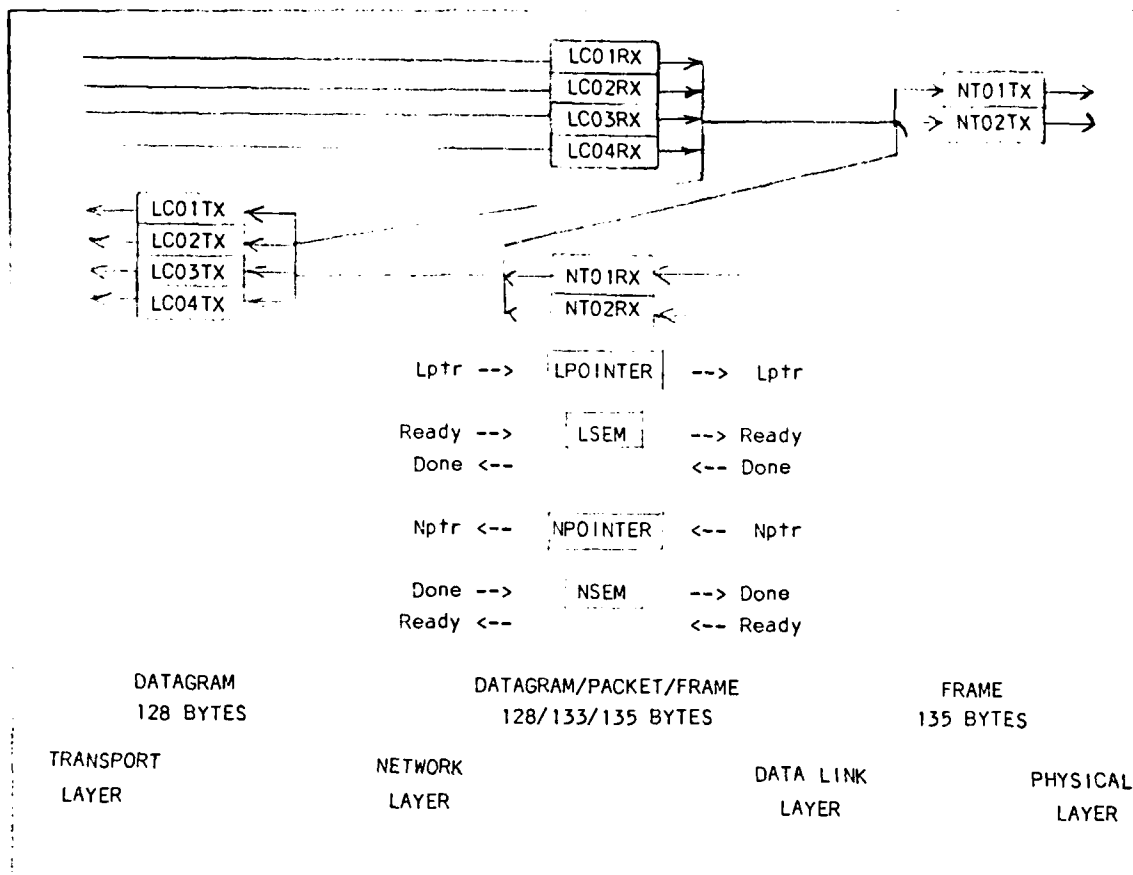


Figure 5-2. UNID II Data Structures and Flow (15:4-5)

The solid lines in the figures above represented data flow between data tables. At the bottom of each figure, the approximate ISO layer corresponding to the data flow was labeled. The data structures passed by each of the tables retained by (15) consisted of datagrams, packets, and frames. The format each of these structures was specified by (93:Append C). The format for the datagram consisted of a minimal implementation of the TCP/IP header within a 128 byte array. Processes operating at layer 4, the transport layer, format the bytes needed in

the TCP/IP header and send the message to the UNID through the hardware RS-232-C interface. While the TCP/IP header used 56 bytes, only 22 bytes were filled out in the IP header and only 6 bytes were filled out in the TCP header. This minimal implementation supported the necessary operations needed by the lower layers within the UNID. The packet format developed by (93) added only five bytes to each datagram. Of these five bytes, one was for a source address and one was for a destination address. The remaining three bytes were unused. This implementation accomplished a primitive network header comparable to the CCITT X.25 recommendation.

Appended to the packet format were two more bytes used to form the partial frame structure. The first byte was an address byte. As implemented by (93) this byte contained the message destination country code and network code for multipoint addressing. The second byte appended to the partial frame format was a sequence byte. This byte used only one bit to implement the sequence number functions. As mentioned above, these two bytes formed the partial frame format.

The complete format structure included a beginning flag byte, the address byte, the sequence byte for LAP B protocol, the packet structure, a two byte checksum, and an ending flag byte. Of these fields, the beginning flag, checksum, and ending flag bytes are handled by the SBC 88/45 hardware. The user, however, must interrogate the hardware status to determine the validity of the frame. This implementation results in a frame structure of 135 bytes (excluding the bytes handled by the hardware), a packet structure of 133 bytes, a datagram structure of 128 bytes, and a user data area of 72 bytes (15:Appendix C).

In the network layer software on the SBC 544, transmit (LC0xTX) tables and receive (LCxRX) tables are datagram size. (The 'x' stands for numbers 1, 2, 3, or 4.) In the data link layer software, the transmit (NT0xTX) tables and receive (NT0xRX) tables are both the partial frame size of 135 bytes. The length of these tables had no analytical basis other than the length of ten datagrams, or in the case of frames, the length of ten information frames. The number of ten structures for each table came about from the hardware memory limitations. When possible, communications between tables used pointers rather than block data transfers. This approach reduced processing time devoted to each message transaction.

The network layer software design implemented corresponds to the upper portions of the ISO layer 3B (Figure 3-2). Layer 3B functions concern the IP header manipulation and channel routing for a multihost environment. The TCP/IP header implemented a variation of the CCITT X.121 internet addressing protocol (93:Append C). The variation applies to the 32 bits allowed for the source and destination addresses in the IP header. The layer 3B software implemented determines the datagram destination from the IP destination address and sends the datagram to the appropriate host. If the host is located on the same UNID, the datagram goes directly to the appropriate receive table. If the datagram is destined for another UNID, a primitive network layer header is constructed and the datagram is sent to the data link transmit table. The primitive software structure charts in figures 5-3, 5-4, and 5-5 show the basic routing algorithm used to send and receive datagrams. More information may be found in (15: 4-5 - 4-17).

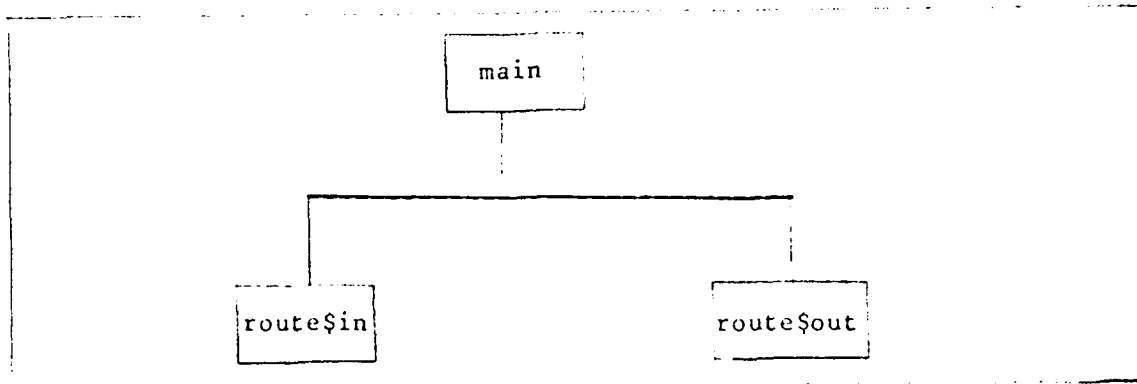


Figure 5-3. Network Layer High Level Structure Chart (15:4-7).

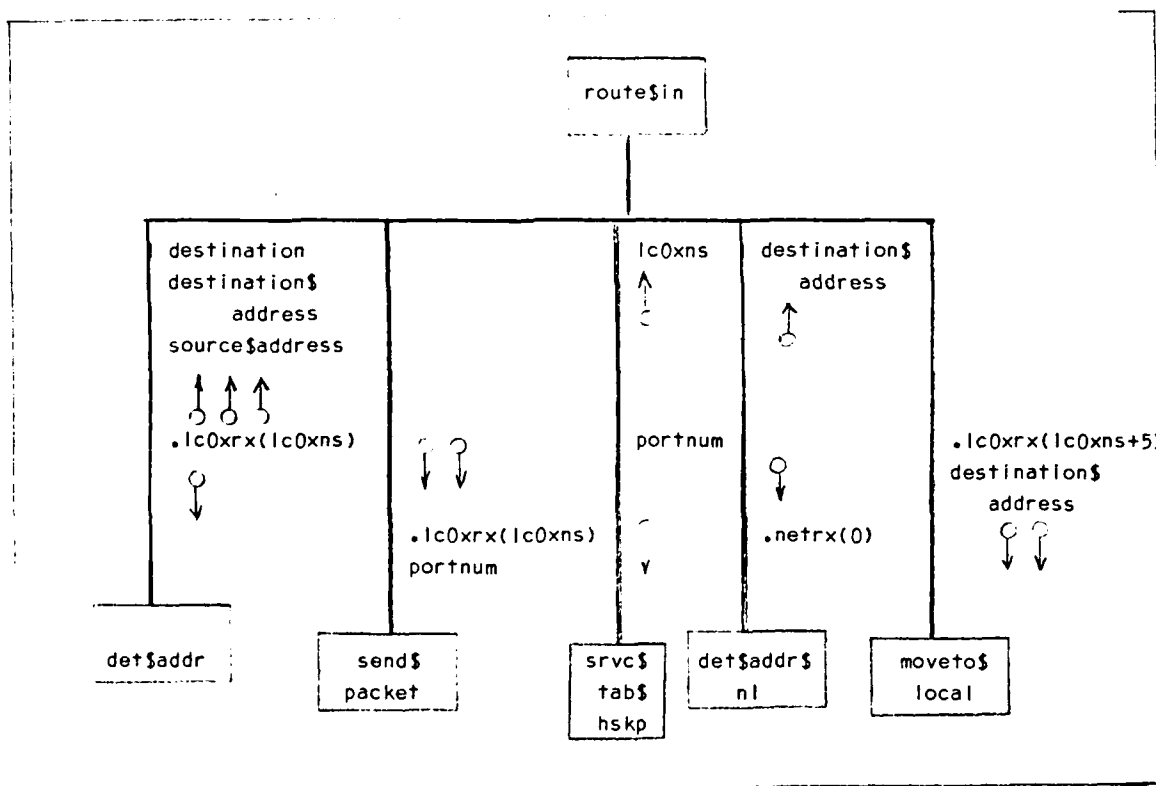


Figure 5-4. Route\$In Procedure Structure Chart (15:4-8).

Figure 5-3 shows the highest level of operations within UNID II software as developed in (15). Figure 5-4 shows the level break down of the "route\$in" procedure. The procedure "route\$in" first determines

the address of a received packet with the procedure `det$addr`. The procedure `send$packet` moves the packet to the correct location within the target receive table. Procedure `"srcv$tab$hskp` updates the receive table pointers. If the packet is determined for a local host from the data link layer (i.e. SBC 88/45) then `"det$addr$nl"` and `"move$to$local"` are called to move a packet to the appropriate host.

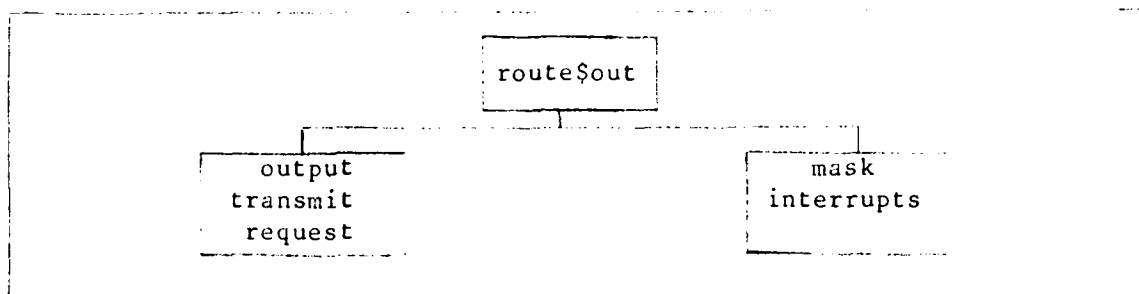


Figure 5-5. `Route$out` Procedure Structure Chart (15:4-11).

Figure 5-5 shows the high level break down of the `"route$out"` procedure. Procedure `"route$out"` sends data to the appropriate receive table through internet driven software handshakes. Figure 5-6 gives the pseudocode for the `"route$out"` procedure. The TR/TA handshake involves four boolean variables which prepare both the sender and the receiver of a datagram for data transmission.

```

disable interrupts
mask receive USART interrupt off
enable interrupts

if datagram available and not sending then
  do
    if TRTA$handshake and (not sending and not receiving) then
      do
        set transmit request true
        set sending true
        send transmit request
      end

      if not TRTA$handshake or (sending and not receiving) then
        do
          set sending true
          disable interrupts
          mask transmit USART interrupt on
          enable interrupts
        end
      end
    end
  end

disable interrupts
mask receive USART interrupt on
enable interrupts

```

Figure 5-6. Route\$Out Procedure Pseudocode  
(15:4-12).

Figure 5-7 shows the TR/TA handshake process when the UNID receives a message from a host. The process for the UNID sending a message to the host is the same with the roles of the UNID is host reversed.

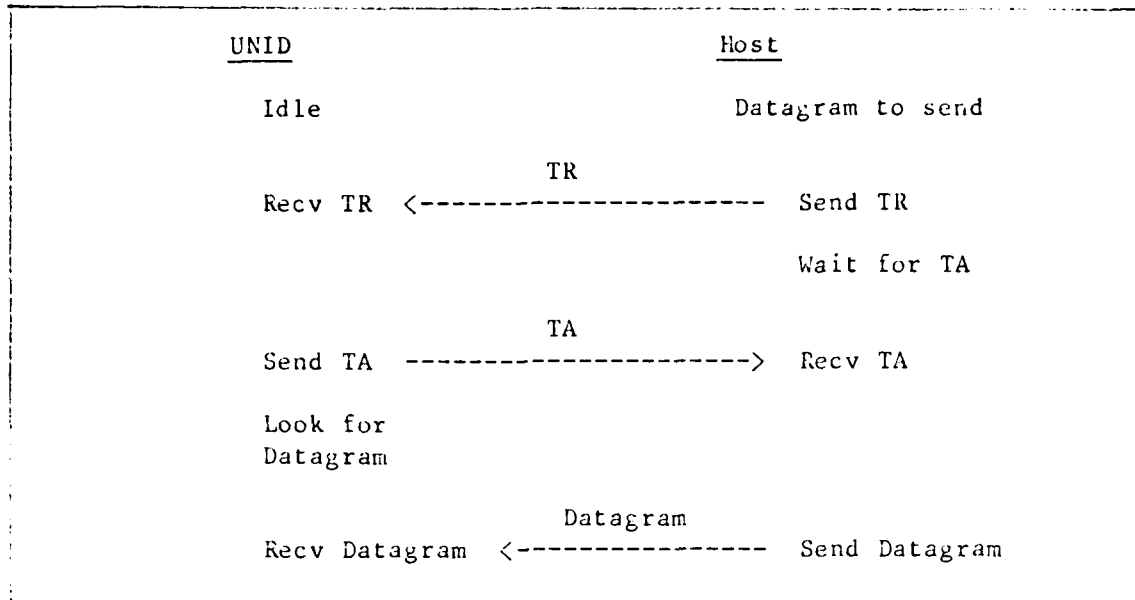


Figure 5-7. UNID/Host Transmit Request/Transmit Acknowledge Handshake. (15:4-13)

More detailed information on the TR/TA handshake procedures may be found in (15:4-13 - 4-17) and in Appendix F. The network layer as completed by (15) is referred to in this document as the network layer 3B and serves primarily as a TCP/IP interface for the UNID and as a multi-host server. Network layer 3A, also referred to as the packet layer, performs the flow control, congestion management, and other functions specifically discussed by paragraphs 4, 5, 6, and 7 in the X.25 recommendation (115:75 - 128). The present condition of network layer 3A is a "straight through" minimal implementation which addresses the manipulation of one byte of the ten byte network layer header. The data link software implementation of previous work (30, 33, 93, 15) allowed only one bit for a sequence number and only processed information (I) frames with limited use of receive ready (RR) frames, and receive not ready (RNR) frames. The structure of the data link layer



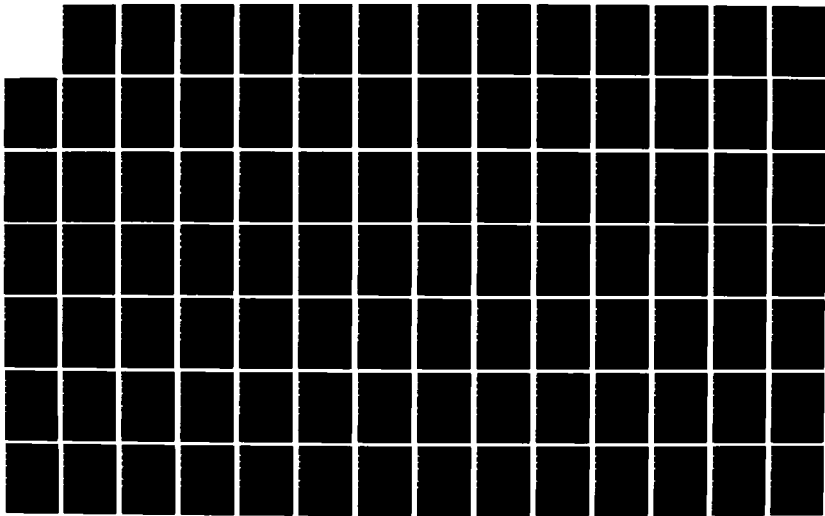
software was similar to the network layer software with "route\$in" and "route\$out" procedure called to move data between receive and transmit tables. The data link layer procedure "route\$in" searched a table (LCNTB) for a datagram destined to the data link layer. When a datagram was present, the "route\$in" procedure filled in the necessary frame header bytes and sent the datagram to the data link layer transmit table. Frames found in the data link receive tables (NTOxRX) destined for the local host were sent to a local transmit table (NTLCTB) where the network layer software continued further processing. Frames found in the data link receive table destined for another UNID were sent to the data link transmit tables (NTOxTX). The procedure "route\$out" handled frames found in the data link transmit tables. Procedure "route\$out" created a software loopback and sent any frames found in the transmit tables back to the receive tables. The two channels, A and B, then had messages circulating from the receive tables to the transmit tables. This simulated the DELNET ring of UNIDs which the data link software accessed. The data link software Childress was able to implement did not use the semaphores necessary to communicate with the network layer software on the SBC 544, nor did the data link software address the four individual receive and transmit tables of the network layer software. The majority of the work in (15) focused on the IP network layer software and programming the SBC 544.

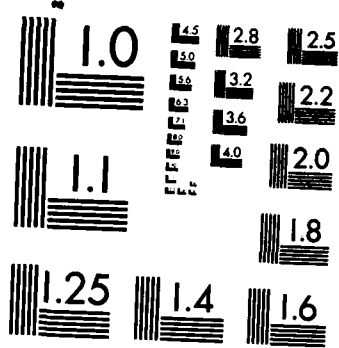
The final design developed by Childress presented the system memory map shown in figure 5-8 for the UNID II.

SBC 544 Memory	Address		System Memory	SBC 88/45 Memory
	Local	System		
				Data Link Layer Software (EPROM)
		FE000		
		14000		
		12000		
SBC 544 Has No Memory Above FFFF		10000	Common Tables, Pointers, Semaphores	
Not Used		C000	0C000	
Network Layer Tables, Pointers, Semaphores		A000		
		8000	08000	
544 Scratchpad RAM	7F00			
		4000	04000	
		2000		Data Link Layer Local Tables
Network Layer Software (EPROM)		0000	00000	

Figure 5-8. UNID II Memory Map (15:4-30).

AD-A164 076      DEVELOPMENT AND IMPLEMENTATION OF THE X25 PROTOCOL FOR      2/3  
THE UNIVERSAL NETW (U) AIR FORCE INST OF TECH  
WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI      M W WEBER  
UNCLASSIFIED      DEC 85 AFIT/GE/ENG/85D-52-VOL-1      F/G 9/2      NL





MICROCOPY RESOLUTION TEST CHART  
NATIONAL BUREAU OF STANDARDS-1963-A

System memory consists of the memory locations accessible by all processors capable of accessing the multibus. Each processor board has its own RAM available to the local CPU. Part or all of the available dual port RAM may be made public to any device in control of the multibus. The tables, pointers, and semaphores common to both the SBC 544 and SBC 88/45 were located at 10000 h in system memory. The memory addresses of the RAM for on board use does not have to match the address used by the system memory. In the case of the SBC 544, the local memory address of the dual port RAM of 8000 h is "mapped" to the system memory location of 10000 h. Note in figure 5-8 that the SBC 544 may only address memory locations from 0 to FFFF h, while the SBC 88/45 has access to an additional 64 k bytes of memory above the uppermost limit of the SBC 544. All the addresses in figure 5-8 are given hexadecimal (h).

#### Development Language Selection

As with previous efforts the language selected for this project was PL/M. The use of PL/M over other available languages removed the necessary step of translating operational software to another language format. In reviewing the language selection, the "C" programming language was strongly considered for two reasons. It is more robust than PL/M in features allowing for more control over data structures and operation at a higher level of abstraction. Though Kernighan and Ritchie may dispute to some degree that "C" is a high level language (76:1), "C" does allow for more control over the basic data structures than PL/M. Finally the wide spread popularity of the "C" programming language (78:1) allow for a greater transportability than is presently shown with PL/M. However,

PL/M and "C" share many common features. Both PL/M and "C" allow modular, self documenting code. Both languages allow manipulation of data structures at the bit level and allow direct access to the hardware ports. Other languages, such as Pascal, lack these features. Despite the above shortcomings experienced by PL/M, the advantages in maintaining PL/M as the "primary" programming language overcame PL/M's inherent limitations. In furthering the X.25 protocol implementations in the existing software, many existing modules needed no changes while others needed only minor alterations. This was the case for most of the IP software developed supporting the SBC 544. The data link layer software required a significant extension to implement CCITT X.25 data link features. Even though PL/M allows the linking of object code modules of different programming languages, the author has had no experience those activities and could find no documentation in describing how separate PL/M and "C" module could be linked together. Therefore, as with past recommendations (15:4-2), "C" may be chosen for a more robust implementation of UNID software, but with current software implementation in PL/M and the familiarity of PL/M over "C", leads the author to the conclusion of maintaining PL/M as the primary programming language.

Some work in 8080/8085 assembly was performed as part of this project. Previous software used an 8080/8086 assembly module for data communications with the UNID software. This software was modified for use with an 8251 USART, the Intel system 210 operating under CP/M, and the Intel system 230, SERIES III operating under ISIS. While the changes were minor in nature, they required the author to gain much experience over two different operating systems and assembly code. The

final result of the program modifications allowed the author to remove all assembly code from the ISIS host software and replace the assembly module with a PL/M module. The basic I/O module remains separate from the main host software to allow modular replacement of the I/O module when the code is transported to other hardware systems.

#### UNID II Data Structures

The data structures from previous work were continued over to the software development of the data link software. While the PL/M supported by the 8086 (PLM86) offers other data types in addition to the PL/M supported by the 8085 (PLM80), facilities for dynamic storage allocation are not documented in user manuals. The use of indexed arrays in the form of FIFO buffers continued as did the use of semaphores to control system memory access. The SBC 88/45 counterpart of the semaphore structures for the SBC 544 were implemented on the SBC 88/45 board. The header format structures used in previous works (15, 93) were altered for compatibility with the CCITT X.25 recommendations. Restructuring the primitive network and data link formats required adding five more bytes to the length of each frame and packet. The first two bytes of the modified frame structure support the data link layer functions within the X.25 protocol. The remaining 10 bytes, which lead the 128 datagram bytes, support the network layer functions. The reader may wish to review Figure 3-7 (3-16) and Figure 3-8 (3-17) for the changes in the format structure of the frame and packet data structures.

The packet header required the addition of five bytes to meet the X.25 datagram format requirements. The format chosen is the fixed

length allowed by X.25 (115:72, Note 2).

The first byte of the packet header contains the General Format Indicator (GFI) in the upper four bits. The GFI is used to determine the type of packet being processed. Packet types include call set-up, clearing, datagram, flow control, interrupt, reset, restart, diagnostic, data, and datagram service signal packets (115:91). Of the GFI formats, only the datagram format is currently used. The lower four bits of the first byte in the packet header are used for the Logical Group Channel Number (LGCN). This number is defined by the user for identifying logical connections between node pairs.

The second byte of the packet header, the Logical Channel Number (LCN) uses the LGCN when an extension of available logical connections is necessary. Presently, the LGCN is initialized to 0 h and not used. The LCN is the address byte 0 (Figure 3-7) of the previous work moved to the LCN byte location and is composed of the source UNID number in the upper four bits and the destination UNID number in the lower four bits. If the UNID's number does not appear in the LCN byte, the packet is looped back into the DELNET ring without further processing.

The third byte in the packet layer header provides the packet sequence number. This sequence byte functions identically to the I frame control byte in second byte of the frame header.

The fourth byte in the packet header denotes the number of "semi octets" or bits in the source and destination address. The upper four bits signify the number of bits in the source address while the lower four bytes signify the number of bits in the destination address. The exact composition of the source and destination addresses is not speci-



fied by the X.25 protocol. In the UNID implementation the values of these bytes defaults to 088 h signifying 8 bits for both the source and destination addresses.

The fifth byte of the packet header has the source UNID, placed in the upper four bits, and the UNID channel number, placed in the lower four bits. The sixth byte provides the destination address. The same format used in source address is used for the sixth byte of the packet header, the destination address.

The seventh byte is used strictly for padding and initialized to 00 h.

The eighth signifies the facility field of two bytes and is set to 02 h. The two byte facility length specifies the minimal facility implementation used in datagram service.

The ninth byte is the facility code byte. The function of this byte concerns strictly the network layer protocol of the X.25 recommendation. The coding of this field is discussed in length in (115:114 - 128). Presently this byte is unused.

The last byte of the packet header, the facility parameter serves to provide additional information when required by the facility code. This byte is also unused at present.

A partial list of some of the functions of the facility bytes follows (115:113 - 128):

- a) closed user groups
- b) bilateral closed user groups
- c) reverse charging
- d) flow control
- e) packet size negotiation
- f) window size negotiation
- g) datagram nondelivery indication

h) datagram confirmation

All functions provided by the facility bytes are optional functions (115:113 - 138) and are not required by any X.25 packet layer services. The 56 bytes of the TCP/IP header which follow the packet header are unaltered from previous works (15, 93). A detailed discussion of the TCP/IP header format used by DELNET may be found in (93: Appendix C).

Network Layer 3B Design

The design as developed by Childress went unmodified except for changes to the packet header format and the transmit interrupt procedures. The design as a whole, is presented here for completeness.

The network layer interfaces to the transport layer through both a hardware and software interface. The software interface consists of the 32 byte IP header format. The IP header format, embedded in the 128 byte datagram, serves to add features not supported by the packet header or transport header. Presently, only eight of the 32 bytes are manipulated in any manner. A detailed description of the IP header format may be found in (93: Appendix C). The hardware interface serves to provide the physical connection between the network layer and the host transport layer. The hardware interface is a sublayer of the network layer and not a true physical layer as would be found in a gateway node (Fig 3-4).

A "three wire" full duplex asynchronous protocol is currently implemented. In lieu of the request-to-send, clear-to-send handshaking found on the "typical" RS-232-C interface, software handshakes as discussed earlier (5-6) are implemented to alert the receive (either DTE, or DCE) of incoming traffic. The four handshake variables were originally developed for interface with the NETOS LSI network and is

considered adequate for the current development of the UNID. Final development will require the full implementation of a synchronous RS-232-C interface supporting the requirements detailed by the CCITT X.21(bis) recommendation (115:44 - 51) Figure 5-9 below provides the simple pseudocode for a synchronous half-duplex RS-232-C interface. Further details may be found in (23, 28:928 - 961).

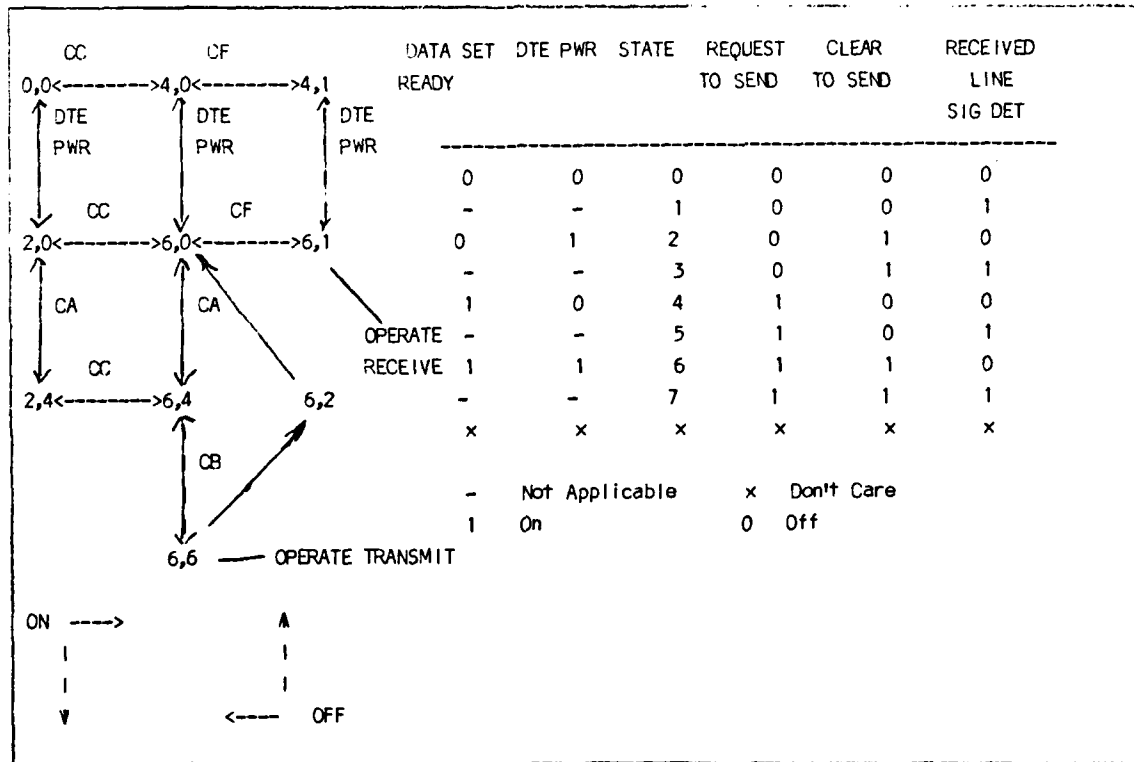


Figure 5-9. Control Lead Sequences for Half Duplex Operation (27:956).

Figure 5-10 shows the allowed state transitions for a full duplex solution using two-point dedicated lines. On transitions are noted by a down or right directed arrow. Off transitions are noted by a up or left directed arrow. Also a (xx) indicates an off transition.

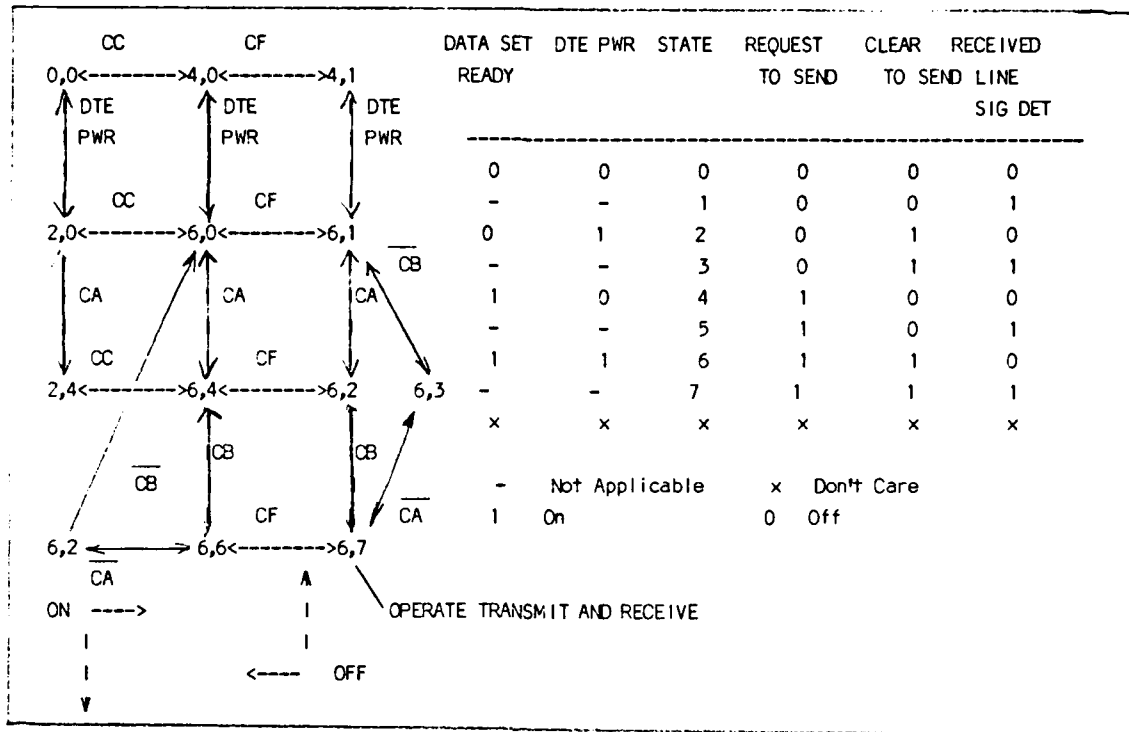


Figure 5-10. Control Lead Sequence for Full Duplex Operation (28:955).

The state diagrams show the various transitions which are allowed for RS-232-C communications. Full duplex operation has the simplest protocol in that the overhead required to change direction is not necessary. Half duplex on the other hand allows the two way circuit to operate at maximum capacity in a given direction (28:935). The host-to-network layer data exchanges involve a fixed datagram length of 128 bytes. Rapid turn around between transmit and receive functions would not be required if the transmission interval was comparatively longer than the turnaround time. Hence, throughput would benefit from the half duplex implementation. This analysis, however, neglects the semaphore polling conducted between the network layer software and the data link software. Moreover, as the baud rate increases from the present 9600

baud to the desired 19.2 k baud, the time duration of each datagram would be cut in half and become less significant when compared to the turnaround time. Further operational analysis would be required to determine the hardware delays encountered by the polling of two data link receive table semaphores by the SBC 544 and the polling of four network table semaphores by the SBC 88/45. Figure 5-11 shows the psuedocode developed for a the control of a simple full duplex RS-232-C interface. Manipulation of the Signal Line Quality Signal for half duplex operation.

```

Initialize interface variables DTR, DSR, RTS, CTS
CALL TURN ON DTR    /* for DTE */
CALL TURN ON RTS    /* for DTE */
do while not DISCONNECT
  do while not CLEAR
    do while ((DTR ON and DSR ON and not (CLEAR or
      DISCONNECT))
      do while (CTS ON and RTS ON)
        TRANSMIT AND RECEIVE DATA
      END
    END
  END
END
END

function DISCONNECT(DTE): DTE sets DTR and RTS OFF

function DISCONNECT(DCE): DCE sets DSR and CTS OFF

function CLEAR(DTE):
  Set RTS OFF, XMIT '0'
  wait for DCE to set RCV to '0'
  if no TIMEOUT then Set XMIT '1'
    AND wait for DCE to set RCV to '1'
  /* CLEAR DTE COMPLETE */
  else set call DISCONNECT

function CLEAR(DCE):
  Set DSR OFF and RCV to '0'a
  wait for DTE to set RTS OFF and XMIT '1'
  if no TIMEOUT then CLEAR COMPLETE
  else call DISCONNECT

```

Figure 5-11. Pseudocode for a Full Duplex DTE

The changes made to the transmit service routines are described in the paragraphs that follow.

The transmit procedure disabled the receive interrupts when data was detected in the transmit table. The transmit procedures would disable the receive interrupt to the channel, transmit a byte, enable the receive interrupt and wait for interrupt processing to detect the next byte in the transmit table. Even though the receive interrupts had

priority over the transmit interrupts, the receive interrupt had to be disabled long enough to transmit a byte which meant it was also disabled long enough to miss receiving a byte. Figure 5-12 shows the original psuedocode used by (15).

```
if ((not trta) or ((txtr and rxta) and ((not rxtr) and
                    (not txta)))) then
  do
    send next character to host
    increment transmitted character count
    increment transmit buffer index
    if number bytes sent >= datagram size then
      do
        * mask transmit interrupt bit off
          reset transmitted character count
          if transmit buffer index >= max index then
            reset
            reset txtr false
            reset rxta false
            reset send false
          end
        end
      clear interrupt

      * - removed when modified for this thesis
```

Figure 5-12. Transmit Interrupt Procedure (15:4-17)

The transmit interrupt procedures were enabled whenever the ROUTE\$OUT procedure was executed. The ROUTE\$OUT procedure provided a means to poll the software handshake variables and enable the interrupt transmit routine when a packet of data was found. The interrupt routines would then update the next-to-send (NS) pointer of the transmit table as each byte is sent to the host.

The modified ROUTE\$OUT procedure psuedocode is given in Figure 5-13. All disable and enable statements of the receive interrupt routines are removed from the procedure. At no point in the data link

software are the receive interrupt routines disabled once initialized.

```
*   disable interrupts
*   mask receive USART interrupt off
*   enable interrupts

if datagram available and not sending then
  do
    if TRTA$handshake and (not sending and not receiving) then
      do
        set transmit request true
        set sending true
        send transmit request
      end

      if not TRTA$handshake or (sending and not receiving) then
        do
          set sending true
          *   disable interrupts
          **  mask transmit USART interrupt on
          *   enable interrupts
        end

      end

    *   disable interrupts
    *   mask receive USART interrupt on
    *   enable interrupts

    * - removed for this thesis
    ** - replaced with CALL SERVICE$TRANS routine
```

Figure 5-13. Route\$Out Procedure Pseudocode

The transmit interrupt routines were changed to polled routines called by the ROUTE\$OUT procedure. All disable and enable statements in procedure ROUTE\$OUT, along with all lines of code between them, were removed and replaced with a single call to polled transmit service routine. Also, the dummy receive and transmit routines for channel one, the SBC 544 monitor channel, were replaced with operational receive and transmit procedures. Hence, channel one is both the SBC monitor port and an



operational UNID port. The final network layer 3B software embedded on the SBC 544 (under filename OP544.SRC) had the two byte data link layer header, the ten byte X.25 packet layer header, four interrupt driven receive channels and four polled transmit channels.

The network layer software in the SBC 544 primarily manipulates the bytes located in the TCP/IP header. Within the ISO reference model, this layer performs the network layer 3B (Figure 3-4) functions. Layer 3B's functions to provide multi-host services, while layer 3A functions to establish packet level management. The present datagram implementation requires none of the call set up procedures implemented in VCS and performs only the minimum layer 3A functions necessary for communications between the network layer 3B and the data link layer. Hence, the network layer 3A development within UNID II has a minimal implementation. As a minimal implementation, the only byte within the packet header manipulated by UNID II software is the GFI/LGCN byte (i.e. the second byte of the packet header). The remaining bytes supply either redundant information or support facilities not implemented.

#### Network Layer 3A Design

The modules RCV\$DATA and RCV\$I\$FRAME of the data link layer software provide the interface to the packet layer protocol as discussed in paragraph 5 of the X.25 recommendation (115:85-90). Many of the functions provided by the packet header have nearly identical processes in the two byte data link header. In consideration of the recommendation given by Tannenbaum (112:245) and of the delay encountered relating to the host--UNID interface, the packet layer was not fully developed. Presently, routing procedures in the simulation software examines the

packet layer LCN byte and sends the packet to either a local host of back on to the network. Additional packet layer functions are not necessary for simple datagram operation. Virtual circuit service (VCS) and permanent virtual circuit service (PVS) do require additional packet layer services, but fall outside of the scope of this thesis (115:92). Further information on VCS and PVS services may be obtained in paragraph 4 of (115:75 - 85). The data structure developed for the packet layer was described in (5-10 - 5-14). Appendix K presents the state diagrams developed from paragraph 5 of (108) and Appendixes A, C, and D.

#### The Datalink Layer Design and Implementation

The previous implementation (15) left the data link layer simulation software with the capability to transmit and receive information frames. The use of data link control mechanisms was limited to receive ready (RR) frames for a correct frame and reject (REJ) frames for incorrect frames. This software only existed as a simulation. The software routed frames between data link transmit and receive tables, between data link receive tables and network layer transmit tables, and between network layer receive tables and data link transmit tables. No hardware initialization routines for the SBC 88/45 were developed, nor did transmit and receive software routines employ the same use of semaphores as did the network layer 3B software.

The references (106, 115) provided the primary background for developing the X.25 design in this thesis. Additional material came from (3, 23, 28, 81, 100, 116) for development of the LAP B data link protocol. From (115) a set of primitive Structured Analysis and Design Technique (SADT) activity diagrams were developed. The SADT diagrams

assisted the conversion of LAP B protocol procedures from (115) to a modular set of processes. From the SADT diagrams, structure charts were developed (Appendix I) which defined the inter-module interfaces between different processes. After several iterations where the SADT's and structure charts were modified to eliminate inconsistencies with LAP B procedures, the PL/M procedures were coded.

The design of the data link software complies with the DELNET requirements presented in Chapter II. The previous implementation of the LAP B procedures was expanded to include all data link command and response frames except the optional asynchronous response mode (SARM) command. Specific details on each frame implemented are discussed later in this chapter.

While the design centered around modifications to the procedures ROUTE\$OUT and ROUTE\$IN from the previous design, other modules were modified to initialize global variables added to the design, accommodate the revised frame and packet structures and to accept variable length frame sizes. Global parameters added include SEND\$STATE\$A and RCV\$STATE\$A. These variables acted as the LAP B V(S) and V(R) variables for channel A. SEND\$STATE\$B and RCV\$STATE\$B acted as the LAP B V(S) and V(R) variables for channel B. Literal constant definitions required changing to reflect the new packet and frame sizes. The constant FRAME\$SIZE was changed to four separate constants (I\$FRAME\$SIZE, S\$FRAME\$SIZE, U\$FRAME\$SIZE, CMDR\$FRAME\$SIZE) to reflect the different frames passed by the data link software. The variable FRAME\$SIZE was then added to the procedures LAB\$TAB\$H\$SKP and SRVC\$TAB\$H\$SKP. The variable FRAME\$SIZE allows access to the indexing distance of the six tables

used by the simulation software. Presently, I frames are a fixed length of 140 bytes. The X.25 recommendation allows for variable information frames. Though not implemented in the software developed at this time, future implementations must determine the length of information frames before the next-to-service (NS) and next-available (NE) pointer may be updated. Finally the semaphore structures used in the SBC 544 for communications with the SBC 88/45 were developed and implemented in data link software.

In general, the program flow of Figure 5-5 (5-8) was changed to the program flow portrayed in Figure 5-14 below. The START\$DM\$MODE of procedure establishes the asynchronous balanced mode of communication across the link. The START\$INFO\$XFER procedure is essentially identical to the main program of Figure 5-5, which calls the procedures ROUTE\$IN and ROUTE\$OUT.

```
main: do
    do while FOREVER is true
        call START$DM$MODE
        call START$INFO$XFER
    end while loop
end main
```

Figure 5-14. Pseudocode for main procedure of data link software

Figure 5-15 below shows in more detail the START\$DM\$MODE. The processing in the procedure assures both channels on a given UNID operate in the asynchronous balanced mode and react to SABM commands as specified by (115:66-67).

```

set SABM mode false in both channels
send message indicating in DM mode
do while not in SABM mode in both channels
  /* Channel A processing */
  if a frame is present in the receive table then
    do
      if not in SABM mode(Channel A) then
        do
          if SABM frame call RCV$SABM(Channel A, P$bit
          else if UA frame call RCV$UA(Channel A, P$bit)
          else send DM response frame
        end
      end
    end
  /* end Channel A processing */

  /* Channel B processing */
  if a frame is present in the receive table then
    do
      if not in SABM mode(Channel B) then
        do
          if SABM frame call RCV$SABM(Channel B, P$bit
          else if UA frame call RCV$UA(Channel B, P$bit)
          else send DM response frame
        end
      end
    end
  /* end Channel B processing */

```

Figure 5-15. Pseudocode for procedure START\$DN\$MODE

Figur5-16 shows the details of the START\$INFO\$XFER procedure. The procedure is essentially the same as the main program for the software previously developed (15).

```

set RNR MODE off for both channels
send message indicating start of procedure
do while in SABM mode for both channels AND FOREVER is true
  call ROUTE$IN
  call ROUTE$OUT
end while loop

```

Figure 5-16. Pseudocode for procedure START\$INFO\$XFER

The module ROUTE\$IN was changed extensively to add procedure calls to the LAP B data link procedures. The calls consisted of send and

receive procedures for receive ready (RR) frames, receive not ready (RNR) frames, reject (REJ) frames, set asynchronous balanced mode (SABM) command frames, disconnect mode (DM) response frames, disconnect (DISC) command frames, unnumbered acknowledgement (UA) response frames, and command reject (CMDR) response frames. The ROUTE\$IN procedure performs both data link and network header evaluations. Datalink processing begins with checking for a frame in the receive data link tables. A valid response continues processing the control byte. Figure 5-17 shows the pseudocode for the ROUTE\$IN data link processing.

```

/*Channel A processing */
control byte = byte 2 of frame
p$bit = bit 5 of control byte
sequence number = bits 8 through 5
if not SABM$MODE$A then
    if control byte = SABM control byte call RCV$SABM(Channel A, P$bit)
    else send DM response
else if control byte = I$frame control byte then
    call RCV$I$FRAME(Channel A, P$bit)
else if control byte = UA control byte then
    call RCV$UA(Channel A, P$bit)
else if control byte = RR control byte then
    call RCV$RR(Channel A, P$bit, sequence number)
else if control byte = RNR control byte then
    call RCV$RNR(Channel A, P$bit, sequence number)
else if control byte = REJ control byte then
    call RCV$REJ(Channel A, P$bit, sequence number)
else if control byte = DISC control byte then
    call RCV$DISC(Channel A, P$bit)
else if control byte = CMDR control byte then
    call RCV$CMDR(Channel A, P$bit)
else if control byte = DM control byte then
    call RCV$DM(Channel A, P$bit)
else service the next to send pointer /* dump the frame */

/*Channel B processing */
control byte = byte 2 of frame
p$bit = bit 5 of control byte
sequence number = bits 8 through 5
if not SABM$MODE$B then
    if control byte = SABM control byte call RCV$SABM(Channel B, P$bit)
    else send DM response
else if control byte = I$frame control byte then
    call RCV$I$FRAME(Channel B, P$bit)
else if control byte = UA control byte then
    call RCV$UA(Channel B, P$bit)
else if control byte = RR control byte then
    call RCV$RR(Channel B, P$bit, sequence number)
else if control byte = RNR control byte then
    call RCV$RNR(Channel B, P$bit, sequence number)
else if control byte = REJ control byte then
    call RCV$REJ(Channel B, P$bit, sequence number)
else if control byte = DISC control byte then
    call RCV$DISC(Channel B, P$BIT)
else if control byte = CMDR control byte then
    call RCV$CMDR(Channel B, P$bit)
else if control byte = DM control byte then
    call RCV$DM(Channel B, P$bit)
else service the next to send pointer /* dump the frame */

```

Figure 5-17. Pseudocode for ROUTE\$IN Data Link Procedure

The network processing consists of an evaluation of the LCN byte for the destination of the frame. Figure 5-18 shows the pseudocode for the ROUTE\$PACKET procedure. This procedure performs destination processing of the LCN byte.

```
if I frame has been received from network layer then
  do
  determine network destination
  if destination for channel A then
    do
    determine if information in transmit buffer A
    if no info frame in transmit buffer then
      do
      build an information frame
      service local buffer pointer
      end
    end
  if destination for channel B then
    do
    determine if information in transmit buffer B
    if no info frame in transmit buffer then
      do
      build an information frame
      service local buffer pointer
      end
    end
  end
end
```

Figure 5-18. ROUTE\$PACKET Destination Processing Pseudocode

The procedure does not load the transmit buffers with an I frame if one already exists. This way the transmit buffer contains no more than one I frame to send at any given time.

The X.25 recommendation specifies the address and control bytes shall be transmitted least significant bit first and the two byte checksum shall be transmitted most significant bit first. The bit order of transmission for the remaining bytes of the frame are not specified.



Throughout this chapter and following chapters, the most significant bit will be the left most bit of the parameter depicted, the least significant bit will be the right most bit in the parameter, and transmission will begin with the left most bit and continue to the most significant bit. As an example, the X.25 recommendation presents the 'A' address for the address byte of the frame as '11000000' binary. This representation is least significant bit first. In this Thesis, the 'A' address byte is given as 03 h or '00000011' binary.

The control byte is first evaluated against the appropriate SABM mode status variable SABM\$MODE\$A or SABM\$MODE\$B. If the UNID has not entered the SABM mode, then only a SABM mode command or UA response to a SABM mode command may be received. All other frames generate a DM response frame and increment the NS pointer which effectively dumps the frame. Once the SABM mode has been entered, all NE pointers, NS pointers, receive variables, and send variables are set to zero and the UNID responds to all data link frames.

Evaluation of the control byte determines the receive sequence number, N(R), and the send sequence number, N(S), and the poll/final (P/F) bit. Only I frames contain the N(S) sequence number. The N(S) sequence number updates the receive state variable V(S). The receive sequence number N(R) contained in I frames, RR frames, RNR frames, and REJ frames updates the send state variable V(S). As RR frames, RNR frames, and REJ frames act as both command or response frames, the address byte must be examined to determine if the frame is an issued command or a response to a command. Should the control byte decode into a non-existent frame type the frame is treated as an invalid frame and

ignored. Currently reception of all CMDR frames simply reset the SABM mode, however, evaluation of the three bytes of status information does allow for more precise handling of rejected frames.

I frames are considered valid when the address byte is a command address for the channel receiving the frame and the receive sequence number is in sequence. Both the receive sequence numbers and the send sequence number consist of three bit modulo 8 fields. The modulo 8 numbers allow up to seven frames to remain outstanding without a duplication of sequence numbers. Current software retains the limitation of one outstanding frame allowed in the data link transmit tables. While further development will wish to increase the number of outstanding frames, (4) indicates a window size of only 3 or 4 frames achieves optimum performance. Passed to the RCV $\S$ I $\S$ FRAME procedure is the received channel number and the sequence number. The RCV $\S$ I $\S$ FRAME procedure, as with all other receive and send procedures, consist of a case structure as shown in figure 5-19.

```

do case (channel)
  CASE Channel A:
    receive sequence number = upper 3 bits of control byte
    send state sequence number = bits 2 - 5 of control byte
    if receive sequence number is in sequence then
      do
        RCV$STATE$A = send state sequence number + 1 mod 8
        update the send state(Channel A) with the
          receive sequence number
        if P$bit = 1 then
          if I frame is next to send
            bits 8 - 5 = receive sequence number
          else call SEND RR(Channel A, P$bit = 1)
        DESTINATION = determine destination of Channel A
        if DESTINATION = network-to-network then
          do
            move frame from Channel A to Channel B
            service next available pointer
          end
        else
          do
            move packet in frame to local host receive
              tables
            service next available pointer
          end
        end Case Channel A
      Case Channel B
        receive sequence number = upper 3 bits of control byte
        send state sequence number = bits 2 - 5 of control byte
        if receive sequence number is in sequence then
          do
            RCV$STATE$B = send state sequence number + 1 mod 8
            update the send state(Channel B) with the
              receive sequence number
            if P$bit = 1 then
              if I frame is next to send
                bits 8 - 5 = receive sequence number
              else call SEND RR(Channel B, P$bit = 1)
            DESTINATION = determine destination of Channel B
            if DESTINATION = network-to-network then
              do
                move frame from Channel B to Channel A
                service next available pointer
              end
            else
              do
                move packet in frame to local host receive
                  tables
                service next available pointer
              end
            end
          end
        end
      end
    end
  end
end

```

Figure 5-19 Pseudocode for RCV\$I\$FRAME.

The evaluation of separate cases for each channel of the data link layer simplifies building in the DCE and DTE properties of the X.25 protocol. For receive I frames, and all other frames, channel A is defined as the DCE device and channel B is defined as the DTE device. The arbitrary definition is also carried out in the physical layer hardware. Channel A sends all command frames with an 'A' address byte and expects an 'A' address byte for all responses (108:65). A 'B' address byte on a frame indicates a command to channel A from the DTE device. Channel B sends all commands with a 'B' address byte and expects a 'B' address byte in response frames. An 'A' address byte on channel B is considered a command frame.

After evaluation of the address byte and the least significant bit of the control byte, the receive sequence number is checked for a receive state sequence number within a single frame window. A valid sequence number updates the receive state variable of the channel receiving the frame and initiates an acknowledgement of the received I frame. An acknowledgment occurs in one of two ways: Should the NS pointer in the receiving channels transmit buffer point to an I frame, then the I frame's receive state variable is updated to the current receive state. Should the NS point indicate a frame other than an I frame, a RR frame with the current N(R) variable is added into the transmit queue. Future implementations of the data link software may also use RR, RNR, or REF frames to update the N(R) variable. Once the receive state variable has been updated, the time-out mechanism is reset. Figure 5-17 shows the possible changes in state variables when an I frame is sent.

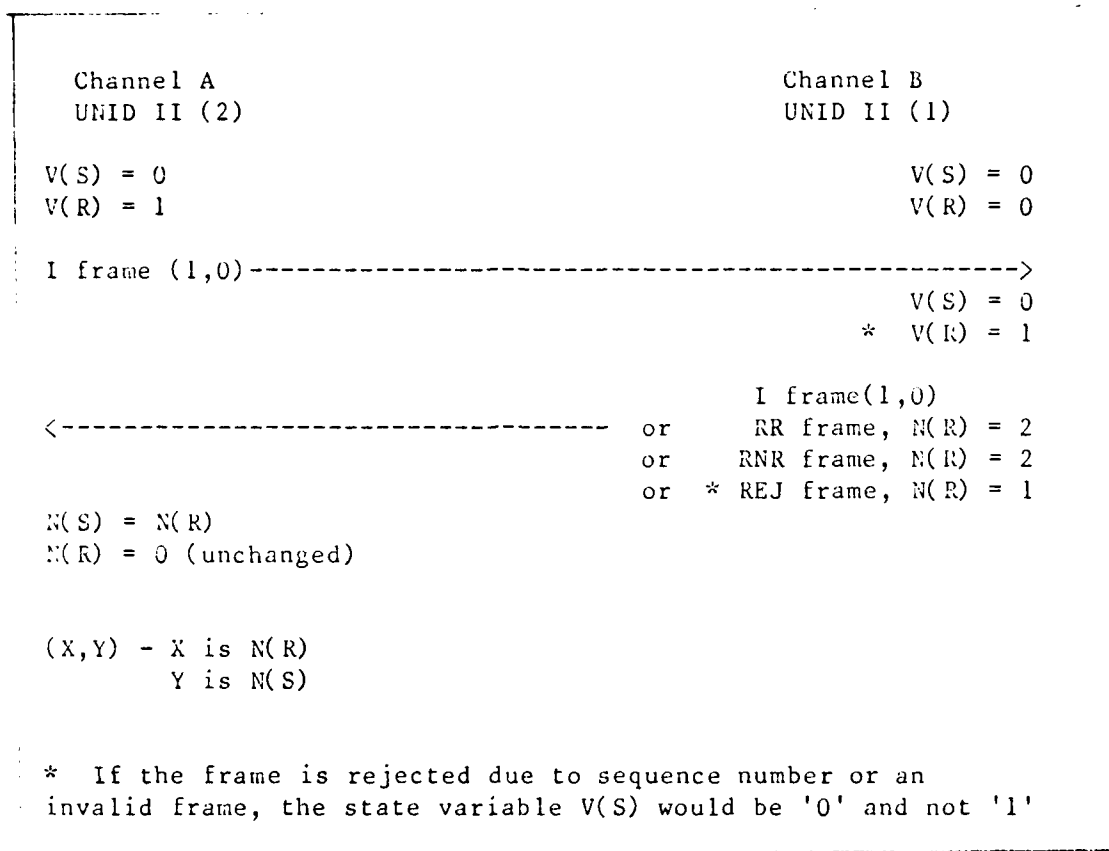


Figure 5-20. Example of a Received I Frame.

Figure 5-20 shows an I frame sent from a UNID (2) to the neighboring UNID (1) inside a DELNET ring. Depending on UNID (1)'s evaluation of the I frame, one of four frames may be sent: another I frame if there is one ready to send back to UNID (2); a RR frame for a successful transfer, but not I frame to return the acknowledgement; a RNR frame for a successful transfer, but a busy condition at UNID (1), and a REJ frame for an unsuccessful transfer. The N(R) variable sent by UNID (1) is then used to up dates the UNID (2) N(S) variable. A reciprocal arrangement occurs when UNID (1) sends a message to UNID (2).

Upon determining the destination of the received I frame, the process becomes a network layer operation. Procedure DET\$DEST\$ONE

(channel A) and DEST\$TWO (channel B) are the network layer procedures which return determine if the frame goes to one of the local hosts or if the frame goes to another UNID. Byte three of the frame (byte 1 of the packet) is evaluated for the UNID destination number. A match between the destination number and the current UNID number moves the frame's packet to the appropriate channel of the SBC 544. A mis-match sends the frame to the opposing channel's transmit table. Channel A will send the frame to channel B to move the frame in a counter clockwise direction around the DELNET, while channel B will move the frame to channel A in clockwise direction around the DELNET. The final processing performed by the RCV\$I\$FRAME procedure updates the NS pointer to the next frame in the receive table. The remaining seven frame processing procedures perform similar, albeit, simpler processing as I frames.

RR frames are implemented only as responses and not as command frames. The frame updates the send state variable of the channel receiving the frame. The RNR mode condition, if set is removed. The RNR mode allows the receive channel time to process existing frames without any new I frames arriving. RR frames, REJ frames, UA frames, and SABM frames remove the RNR mode condition. In addition to resetting the RNR mode condition, the time-out mechanism is reset and the NS pointer is incremented to the next frame in the receive table.

RNR frames function identically as RR frames except, instead of removing the RNR mode condition, the RNR mode condition is set. At present, RNR frames are not sent as the single frame window allows reception of only one I frame at a time. Future implementations may

chose to count the number of outstanding I frames received and upon passing a predetermined threshold, issue a RNR frame instead of the RR frame acknowledgement.

REJ frames function similar to RR frames except, the receive state variable returned indicates the next in sequence number expected was not the one receive. The REJ frame functions as a negative acknowledgment and does not update the time-out mechanism as implemented. When more than the maximum allowed retransmissions occur, a CMDR frame is sent which resets the link in the SABM mode.

DM frames are sent as responses while the channel is not in the SABM mode of operation. The received frame is simply ignored.

SABM command frames function to set the channel's NS, NE, send state variable, and receive state variable to zero. The SABM command also removes the RNR mode condition and effectively dumps any frames awaiting processing in either the transmit or receive tables.

DISC frames sever the communication link. In a virtual circuit service link, the link would go to an inactive state, however in datagram service, the link merely reverts back to the DM condition. As the set asynchronous response mode (SARM) is not used or implemented in current software, the link simply tries to reset the SABM mode.

UA frames are set whenever a SABM or DISC command frame is received. The RCV\$UA frame procedure retrieves the last U command frame received and initiates the appropriate processing.

CMDR frames provide three bytes of status information. At present no analysis of the status bytes is performed to determine an optimal error correction procedure. Upon reception of a CMDR frame the link is

reset using a SABM command.

The procedure ROUTE\$OUT first evaluates the control byte of the frame indicated by the NS pointer. The length of the frame (I frame, S/U frame, CMDR frame) is evaluated and passed to the transmit serving routine. No acknowledgement is awaited for S or U frames. All S frames are sent as responses, not commands, and all U frame commands are not acted upon until a UA response is received. I frames force the procedure to invoke a time delay mechanism allowing up to MAX\$RETRANS number of retransmissions before an error is signalled. The variables MAX\$RETRANS\$A and MAX\$RETRANS\$B are considered systems parameter and may be changed to provide optimum performance.

#### Conclusion

This chapter briefly presented the previous design (15) and then detailed the subsequent software design changes, modifications, and further implementations. First, the programming language selection of PL/M was presented. Next, the data structures, including the frame and packet header formats were reviewed. Changes to the frame and packet headers presented in detail. Modifications to the SBC 544 network layer (layer 3B) software was presented. A brief presentation of the the packet layer (layer 3A) was developed. The remainder of the chapter concentrated to the changes and additions made to the data link software ROUTE\$IN and ROUTE\$OUT procedures. Procedure ROUTE\$IN now calls specialized procedures to handle the LAP B functions. Procedure ROUTE\$OUT now handles all types of LAP B frames. The following chapter, Chapter VI, presents the integration and validation phase of the software development implemented during this thesis effort.



CHAPTER VI  
SOFTWARE INTEGRATION AND VALIDATION

Introduction

This chapter presents the test criteria developed to integrate software into the UNID II design and validate the operation of the design. The first section discusses the test philosophy used for the validation process. The following section discusses the modification to host software needed to test the UNID II. The next section outlines the test plan. The remaining sections detail the test plan. Each test phase first discusses the purpose of the test and then presents the results. The chapter concludes with a brief summary of the tests performed.

Test Philosophy

Software validation methodologies may be grouped into two categories: validation and verification. In this thesis effort validation is interpreted as an examination of software for a given finite set of conditions. Verification, in contrast, means a mathematical 'proof' of the correctness of the software design. Often, and particularly in cases involving long detailed programs, test verification becomes an impractical burden on those resources limited in a software development project such as time and finances. This Thesis effort considers the validation approach not only pragmatic, but the only approach which adequately balances available resources against the scope originally defined for the software development.

This design used the conventional top-down approach to overall project development. Each task was partitioned based upon its relation to the ISO reference model. For the network layer 3A and the data link layer the abstraction at each level was then partitioned using either existing state diagrams as given by the X.25 recommendation (115: Appendixes A, B, C, D) and the RS-232-C standard (24) or through SADT diagrams developed from the X.25 recommendation. From the abstractions, whether state diagrams or SADTs, structure charts were developed and then pseudocode procedures were written for each module. As common in software development cycles, the code implementation uncovered shortcomings of the design and several interactions between detailed design and implementation were necessary.

For the network layer 3B and the physical layer, a simplified approach was taken. The network layer 3B (TCP/IP protocol layer) had previously been validated (15: Chapter V). However, the software was configured for equipment no longer available for UNID II development. In lieu, of redesigning all the host software used to test the UNID, structure charts were developed and I/O dependent procedures were modified for equipment available within the Computer/Communications Laboratory (16). The physical layer design was treated as a single problem: How were the UNIDs to interconnect within DELNET? While several possibilities exist, it became evident from the data link software design that the UNID would exhibit the unusual characteristic of having both a DCE and DTE high speed interface channel. Further research (16) provided collaborative support for the physical layer integration.

#### Integration / Validation Tools

Test tools implement the methodology outlined by the test philosophy. In the case of UNID II, path testing methodology provided the necessary details to validate software. Path testing marked critical areas of the program flow. The tools used to implement path testing consisted of diagnostic test messages, a host software program, a monitor program, and finally a piece of test equipment called a protocol analyzer.

The primary diagnostic aid used in the past works (15, 84, 93) and the present work remains insertion of diagnostic test messages with the development software. The messages primarily show path direction, but may include error conditions or update status on key program variables. Most messages demonstrate successful completion of a process by their placement at strategic locations within modules. Some messages, on the other hand indicate areas where erroneous paths may be taken or faulty conditions occur. Some messages display current values of key variables so as to document proper progression of counters, pointers, and sequence numbers.

The display diagnostic messages requires a terminal attached to the device executing the software. As messages are displayed on the terminal, the observer analyzes the path flow. From this analysis, the observer is assured the correct path was taken at the appropriate time. The false occurrence of a diagnostic display or the occurrence of an inappropriate sequence of diagnostic messages informs the observer of improper software path flow. A trace of diagnostic messages then locates the departure from the correct path. Failure of a diagnostic message display, by itself may indicate one of several existing condi-

tions. The problem may exist in the hardware apparatus or software associated with the diagnostic message. As mentioned by (15:5-4):

"The lack of a display message is not conclusive proof that a problem exists in the software as other conditions may exist to prevent the display of the message. The observer needs to look further at the conditions of the test or even generate other tests or diagnostics to determine if the software is at fault."

From the past thesis work (15), a CPM based host program was designed for communications with UNID II. The host program sent messages to UNID II, received messages from UNID II, displayed portions of the TCP/IP header, and effectively simulated the transport and internet protocol required by a fully functional host attached to one of the four UNID II ports. Use of the host test program allowed testing of UNID in an environment simulating expected operational conditions.

A monitor program developed for the SBC 86/12A (15) was used to examine UNID system memory. Placing the SBC 86/12A in the card cage allowed examination of the contents of the multibus system memory. The system memory contains the four local host receive tables and two network receive tables, as well as the pointers and semaphores to those tables. All components involved in communications between SBCs must have precise locations known to each SBC. Close examination of the declare statements for the operational SBC 544 and SBC 88/45 software (Appen H) will note the careful assignment of all pointers, semaphores, and receive tables in contiguous memory locations. Moreover, the centralized memory locations allows the SBC 86/12A monitor to access the data variables in one localized portion of memory. The SBC 86/12A

monitor found use where insertion of diagnostic messages proved inadequate or a large number of data variables required display. This tool provided a 'snap shot' window to the data exchange processes occurring between the SBC 544 and SBC 88/45. The value of this trouble shooting aid can not be over stated. For operational details of the monitor software embedded on the SBC 86/12A refer to (isbc 957 INTELLEC - isbc 86/12A Interface and Execution Package USER's GUIDE).

The last diagnostic tool used was a protocol analyzer, specifically, a HP 4951A. This device serves to monitor communications between a DCE and DTE device or simulate either DCE or DTE interactions using commonly implemented protocols. A simplified modular programming language embedded on the HP 4951A allows specific tests tailored to user needs. The HP 4951A supports the X.25 protocol as well as HDLC, SDLC, and user developed protocols. The primary use of this device was to observe the actual data hardware and software handshakes made between the host device and UNID II. The protocol analyzer will serve an increasing role as a test diagnostic/test validation tool as the UNID II protocol is extended (34).

#### Test Outline

Testing was accomplished in four phases. Phase One testing consists of procedure software testing. Individual procedures are examined for operational suitability and completeness. Phase Two testing consists of simulation software programs. The simulation software provides a realistic environment for software modules and fully exercises specific tasks before their placement on target hardware. Phase Three testing consists of an evaluation of operational software for the SBC 544

and the SBC 88/45. The evaluation concerns the operational testing of the SBCs as individual units with additional software simulating the presence of the missing SBC. Phase Four Testing consists of integrating the SBC 544 and SBC 88/45 software and hardware components. The tests examine the data exchanges between SBCs.

#### Phase One Testing

Modules from previous software development (15, 84, 93) were assumed to operate correctly as stated in introductory remarks to this Thesis (Table 1-1). Given the conditions and limitations as stated in (15) no module fault was ever attributed to the previous works. An inconsistency, however, was noted between the simulation and operational software. The simulation software reversed the bit order of the most significant bits and least significant bits in the TCP/IP header fields. While this in no way impacted the actual operation of the software, it did present interpretation problems. After investigation, the operation software was determined to have the correct implementation (14, 38, 39, 40, 41, 42, 43, 44).

The modules developed for the LAP B data link protocol were tested either through individual drivers, or as groups of similar processes. A 'garbage collection' technique was used to accumulate functional procedures. Procedures not under test were either not present in the driver or commented out. Once a procedure was considered validated under Phase I testing, it was considered usable for validating other procedures still in Phase I testing.

In PL/M, modules are considered a set of one or more procedures properly identified with a module name. A program in PL/M must con-

sists of one module titled "main" and may have several modules with names other than "main". Variables and procedure declarations between the main module and subordinate modules must be resolved through the use of PUBLIC and EXTERNAL attributes to the declarations. Once created, modules are compiled separately from one another. Hence, operationally validated code need not be recompiled with every minor change. The software developed for the data link layer simulation eventually required three separate program modules. The software quickly out grew the PLM86 compiler's symbol table during the development process. The software had to be divided into four separate modules (main, LAPB0, LAPB1, PCKT) to resolve dynamic storage allocation errors. Presently the modules main and LAPB are near the maximum size permissible in the PLM86 compiler's symbol table. Further additions to the data link software should be accomplished in a separate program module.

#### Phase Two Testing

Both the SBC 544 and SBC 88/45 used simulation software to validate procedural aspects of each board's target software. The simulation software was developed on the Intel System III under the ISIS operating system. The simulation software lacked initialization routines for programmable devices and the receive and transmit procedures for the communication ports found on the SBC 544 and SBC 88/45. The simulation software allowed testing and validation of key operational software procedures in an environment facilitating change and modification. For example, the compilation of a 1000 line module normally takes 10 - 15 minutes (all compiling is done on the 8085 processor for both PLM80 and PLM86). Once compiled the time to change the file format to a CP/M

hex file, program and install the EPROMs on the target system takes approximately 30 minutes or more. Simulation software, on the other hand, merely requires program execution with immediate results.

Once the operational software was validated through simulation, datagrams, packets, and frames were known to correctly pass through the individual nodal layers. Also, datagrams, packets, and frame headers were all shown to be correctly interpreted and executed. Moreover, each message unit, whether datagram, packet or frame was known to be correctly routed to the intended destination. Even though the simulation software required further integration with programmable device initialization routines, I/O drivers, and memory allocation assignments, the simulation process installed a high degree of confidence in the suitability and operability of the target software.

Previous thesis efforts (84, 93) used the In-Circuit Emulator (ICE) for software validation. The ICE system required installation of the operational software on the target system before testing could begin. The ICE system did not provide ease of access to the executing software as did the Intel 230 under the ISIS OS. Moreover, the ICE system was more suited to trace hardware failures than tracing incorrect path flow, or monitoring exchanges of data. Altogether, the ICE system was considered inappropriate for the present level of UNID II development.

The general construction for the SBC 544 and SBC 88/45 software was the same. A test message was used as the data section of the TCP/IP header. The TCP/IP header was filled with dummy values, except for data fields which effected the source and destination addressing. These values were filled out through user responses to questions posed by the



software. The user's response routed the TCP/IP, packet, and frame headers in the simulation software. Global assignments within the simulation software established the UNID as the second UNID of a DELNET ring of three UNIDs. Messages sent to UNID number 2 were looped back through software to one of the four host channels on the UNID under test. Messages sent to the channel originating the message were displayed on the terminal. Messages sent to the other three channels could be displayed by another host or simply by an attached terminal.

The SBC 544 simulation software had been previously shown to function correctly. The changes to the packet header were made on the simulation software prior to modification of the operational software. Once correct routing and processing was validated by the simulation software, the operational software was modified. Figure 6-1 shows graphically the operation of the simulation software for the SBC 544.

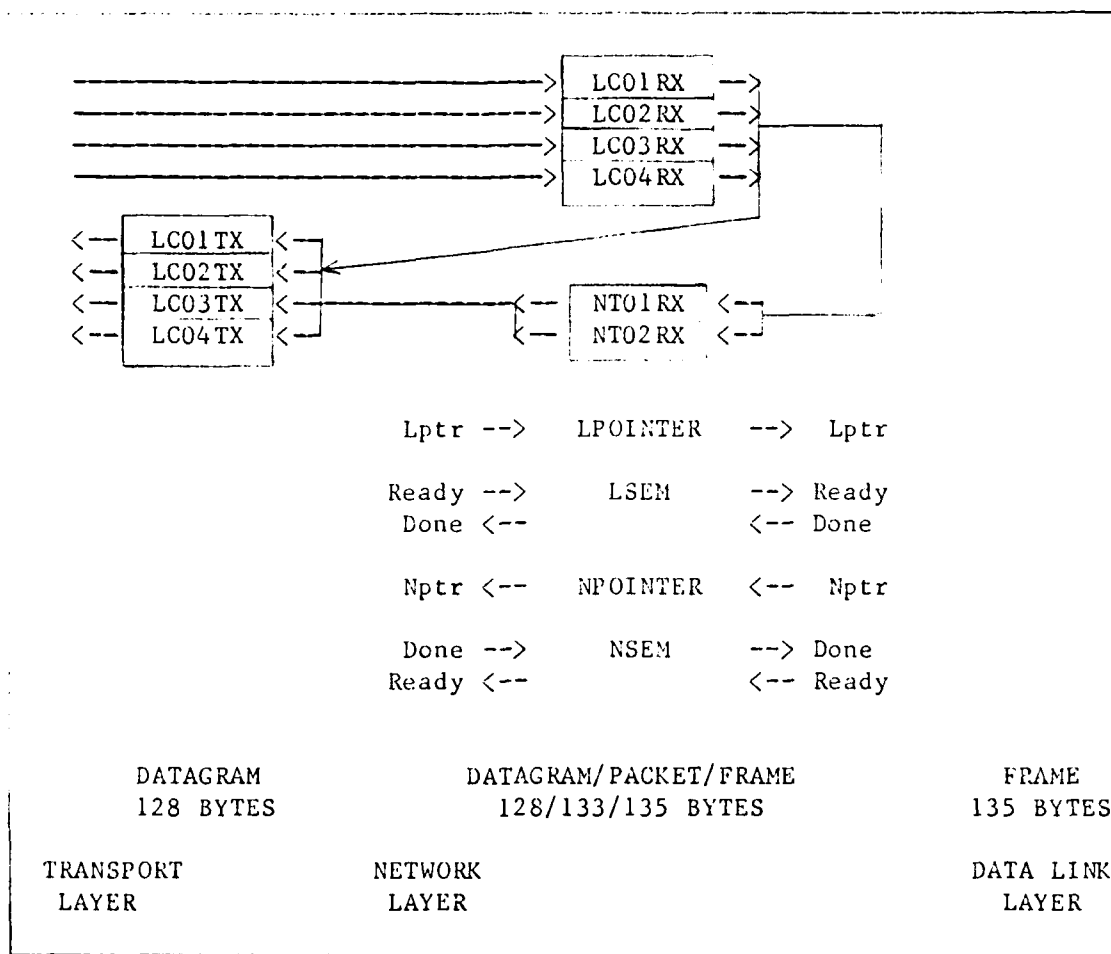


Figure 6-1. Network Layer Simulation Data Structure and Flow (15:5-7)

The simulation software loads 0 to 9 test messages to the LC01RX receive table on the host. The user could then select destination network code (UNID number) and the host code. A '2' network code sends the test message to one of the four transmit tables. Host codes range from '0' to 'FF' h. Each channel's host codes consists of 0 to 3F h for channel one, 40 h to 7F h for channel two, 80 h to BF h for channel three, and C0 h to FF h for channel four. The Series III monitor displayed the test message sent and the message received after routing by the simulation software. Bytes 12 to 19 of the TCP/IP header were

displayed to indicated proper source and destination addresses. The actual test message consisted of a field of 72 bytes loaded into the TCP/IP data section. Within that field, a modulo 10 number identified each specific message sent through the simulation software. As indicated by Figure 6-1, once a message was routed to the data link software receive tables, a software loop returned the message to the local host transmit tables (LCOxTX). Once received by the host receive table, the message was displayed on the system monitor.

The SBC 88/45 simulation functioned similarly to the SBC 544 simulation software. Test messages with modulo ten identifiers were loaded into a dummy network receive table (LCNTTB). The network would examine this table for a message and, if present, load the message on to one of two transmit tables (NT01TX or NT02TX). A simple routing algorithm based on a comparison of the destination address with the current UNID number, sent the message to the appropriate data link transmit table. The routing algorithm required the UNIDs sequentially numbered within the DELNET ring. The largest UNID network code must then connect to the UNID '1' network code. The routing algorithm would then determine which number was closest to the current UNID network code. When a UNID II with network code '0' is developed, it will naturally serve as the smallest network code in place of the UNID '1' network code. Figure 6-2 shows the data link simulation software path flow.

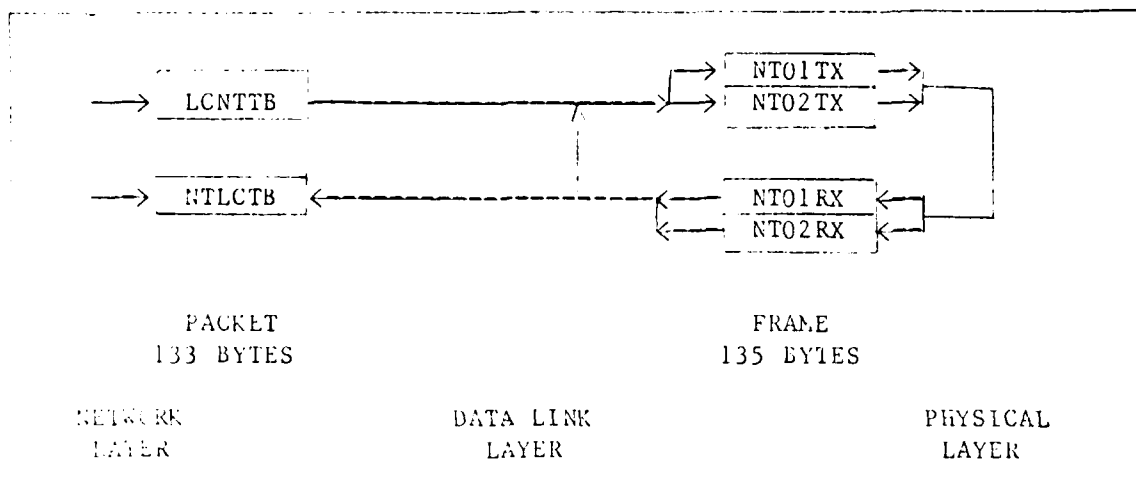


Figure 6-2 Datalink Layer Simulation and Data Structure Flow  
(15:5-8)

To effectively simulate traffic on the DELNET, a software loop sent messages sent from the transmit channels to the receive tables. The receive tables then sent frames destined for the local host to a single table (NTLCTB). Frames destined for another UNID were routed out the appropriate transmit table. Figure 6-3 gives the pseudocode used to query the user for routing messages.

```

send request to load test message
read response
if 'yes' then
  do
    ask for which UNID message is destined
    read response
    if valid response then
      do
        ask how many datagrams to load
        read number
        if valid number then load number requested
      end
    end
  end

ask to load any control frames
read response
if 'yes' then
  do
    give menu selection
    read number
    do case of number read
      Case 0: send CMDR to both channels
      Case 1: send RR to both channels
      Case 2: send RNN to both channels
      Case 3: send REJ to both channels
      Case 4: send UA to both channels
      Case 5: send DISC to both channels
      Case 6: send DM to both channels
      Case 7: send SABM to both channels
    end case
  end loading control frames
end

```

Figure 6-3 Pseudocode for procedure READ\$LINE

Unlike the simulation software for the SBC 544, the SBC 88/45 required extensive changes to process the LAP B procedures. The software loop back procedures had to process channel A using DCE software protocol and process channel B using DTE software protocol. Previous works had failed to note the asymmetrical characteristics of the DCE and DTE data exchanges extended beyond the physical layer into both the data link and packet layers of the X.25 recommendation. Appropriate modifications were required to the frame and packet structure, and the

procedures ROUTE\$IN AND ROUTE\$OUT which evoked the LAP B software procedures. The revised simulation software for the 88/45 significantly aided the refinement of the final LAP B procedures. The final simulation software displayed the test messages sent, the message received, indicated critical flow paths encountered, indicated placement of NS and NE pointers in the six tables used for the simulation, and created a file (FILE.OUT) for a permanent recording of the sequence of events. To fully validate each frame's receive and send processing, the simulation menu was expanded to allow the user to send any of the S or U frames. the hard copy print out of the output file allowed detailed review of an entire session of exchanges between all six tables. The output file procedure may be disabled by changing the literal declaration of FILE\$OUT from 'Offh" to '0' as would be done in a similar 'C' program with a preprocessor define statement.

The data link software was evaluated against the performance requirements given in paragraph 2 of the CCITT X.25 Recommendations (115). The testing made extensive use of the file created by the simulation program, FILE.OUT. FILE.OUT records all messages sent to the system console by the simulation program. Upon invoking the simulation program, FILE.OUT reinitializes itself to an emptyfile, hence, data never accumulates from previous runs of the simulation.

Testing of the data link simulation in general consisted of loading designated test messages into a table simulating one of the four SBC 544 receive tables. Messages were then sent to one of the data link transmit tables. The software allows specifying which transmit table through a query in the READLINE procedure. The procedure READLINE

asked for a UNID, "1" or "3", to send the message. A response of "1" results in the software routing algorithm determining the shortest distance from the current UNID, UNID (2), to UNID (1) as through channel A. Likewise, a response of "3" results in the software routing algorithm determining the shortest distance from the UNID (2) to UNID (3) as through channel B. For simulation purposes, the message is looped back to the receive channel of the UNID. Thus, a message from transmit channel A returns to channel B; a message from transmit channel B returns to channel A. Frames received by a channel are then displayed by the READTAB procedure.

I frames destined for another UNID (such as UNID (1) or UNID (3) in the simulation software) would normally be evaluated, sent to the appropriate transmitting channel, and then sent out into the network. To avoid the network table from accumulating I frames, the simulation software dumps the I frames after determining the messages destination. This was done by simply not moving the frame to the designated transmit table.

I frame protocol was tested by placing various combinations of test messages in either or both channels and noting whether

- 1) the frame was ever received by a transmit table
- 2) the sequence numbers were being updated properly.

The testing of I frame routing brought out the crucial synchronization aspects not covered by the X.25 protocol. In general, whenever, a frame, any frame, has been manipulated the next-to-service (NS) pointer and next-available (NE) pointer "must" be properly placed for

manipulation by "any" following procedure. If the NS pointer for any of the tables manipulated does not always point to the next frame to service in the queue, conditions for program lockout exist. Likewise, if the NE pointer for any table does not always point to the next available location to load a frame in the queue, conditions for program lockout exist. As implemented "every" procedure in the data link protocol to assumes the NS and NE pointers are properly placed when the procedure is invoked. As such, the NS and NE pointers must be properly positioned for every possible occurrence within the procedure.

The DISC and SABM frames were tested for their ability to reset the various variables and pointers under their control. Both of these frames tested successfully.

CMDR and DM frames tested successfully to the extent of their implementation. This implementation was briefly described in Chapter V.

RR, RNR, and REJ frames were tested as responses only. Both RR and REJ frames were found to function properly. RNR frames would place the link in the "receive-not-ready" mode, but the condition could not be recovered without a link reset (DISC frame) or the SABM command frame. Neither The RR nor the REJ frames would successfully return the link to normal operation. This problem remains for further investigation.

#### Phase Three Testing

Phase Three testing required modifying the previously developed host program for operation on the available resources within the Computer/Communications Laboratory. Two separate systems were implemented as hosts for the UNID. The previously developed (15) program (SBS.ASM) served as the basis of three separate host I/O driver modules



for the main PL/M module. HOST1.ASM operated on the Intel 210 under the CP/M operating system. HOST1.ASM allowed configuration of the baudrate, parity bits, and stop bits for the 8251 USART used on the Intel 210. HOST2.ASM operated on the Intel 230 under the ISIS operating system. The module operated under the default conditions of 9600 baudrate, one stop bit, and no parity checking. No menu selection was included as it was not considered crucial for the current stage of software development. HOST3.SRC was simply the HOST2.ASM program written completely in PL/M. HOST3.SRC simplifies the development process for any future modifications to the host I/O driver routines through the use of a high level programming language (PL/M) without the usual requirement of writing assembly code. All three host modules are included in Appendix K to show the simple process of converting an assembly written module to one written entirely in PL/M. Further development of any I/O driver routines for UNID II interfaces should carefully consider using PL/M in lieu of assembly code.

Once the host software was modified and verified in operational tests with the operational SBC 544 UNID II code previously developed (15), the transmit interrupt routines were modified for consistency with the CCITT X.25 recommendation. To embed the software on the target system (the SBC 544), two 2732A ERPOMs required programming. The process required moving an ISIS formatted object code file to the RAM memory of a Bytek EPROM programmer. Intermediate steps in the process required changing ISIS hexadecimal formatted files to CP/M hexadecimal formatted files. Specific details of the conversion process may be found in Appendix D.

The operational tests used an UNID II with the original software, a second UNID II with the polled transmit routines, and the HP 4951A protocol analyzer. The protocol analyzer's program had a timer start when the last two bytes of a test message occurred and stop when the first four bytes of the test message arrived. The Intel 230 host software (ISHOST) was modified to send up to nine datagrams continuously to the UNID (DPHOST) without waiting for a response. The DPHOST software had a menu selectable delay from 0 to 2.56 seconds between datagrams. When eight messages were sent, the protocol analyzer accumulated the time for seven separate delays. The accumulated time divided by seven represented the average delay time between datagrams. A H19 terminal on the UNID II received the messages. Errors in reception could be noted from careful examination of the H19 display or through examination of the actual memory contents using the SBC 86/12A monitor. The SBC 86/12A allowed precise location of which bytes were lost when receive error occurred. The H19 display, on the other hand, allowed a quick identification of improperly received datagrams. Both techniques were employed in the final testing. Figure 6-4 shows graphically the conditions measured by the test set-up and Figure 6-5 shows the test configuration used.

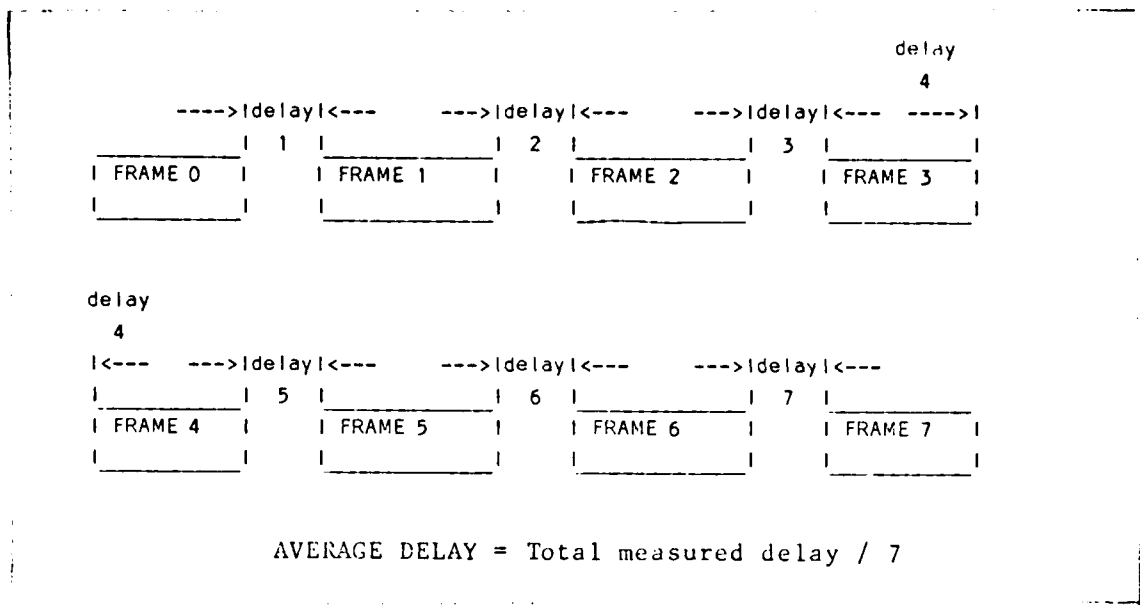


Figure 6-4. Frame Delay Test.

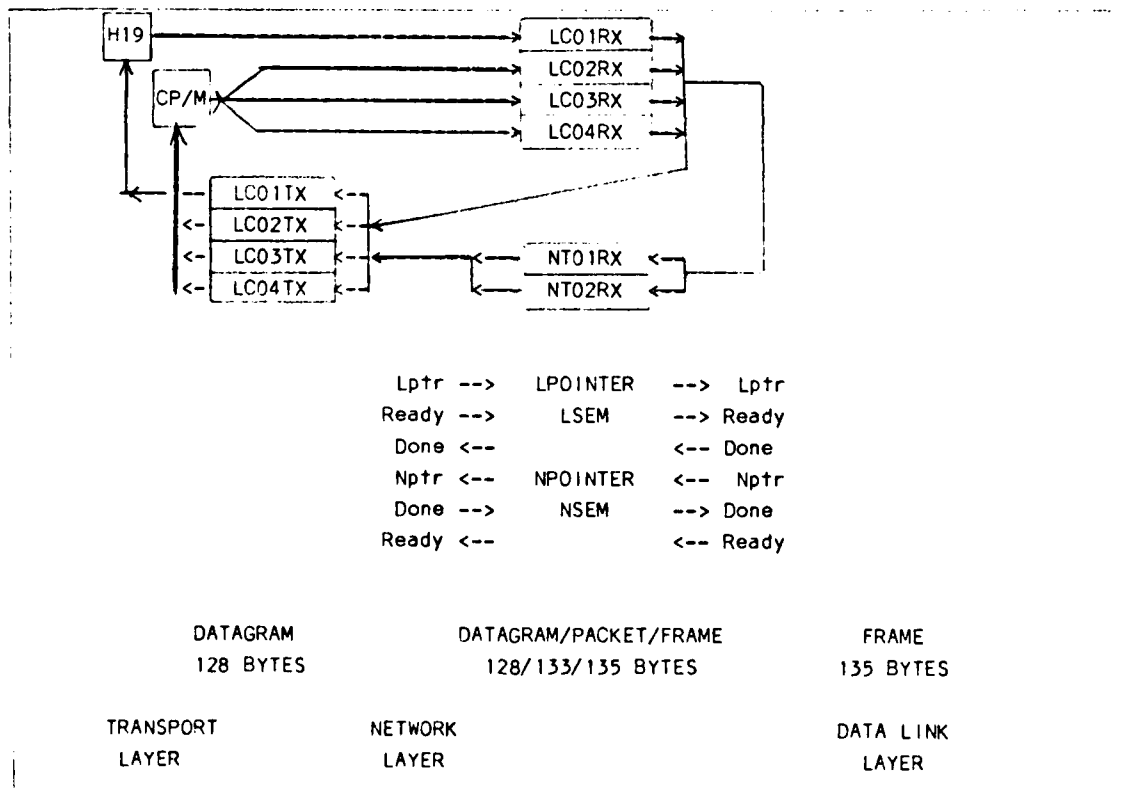


Figure 6-5. Network Layer Simulation with the CP/M System and H19 (15:Figure 5-3)

The final tests show the original software lost bytes when frames approached within 290 ms of one another. In all cases less than 290 ms, one or more bytes were lost. The polled software showed no errors during testing with the protocol analyzer and the DPHOST software. However, the test software could only reduce the delay between datagrams to 110 ms. At 9600 bits per second, this was approximately 141 bytes or slightly more than one packet length. As the only change made to the original software on the SBC 544 was in the transmit routines, the transmit interrupt routines were determined to be the source of the errors detected. The interrupt transmit routines contained disabled statements for the receive interrupts. Since the polled transmit routines removed all disabled statements for the receive interrupts in the host software, the transmit routines are no longer considered a contributing factor to lost receive packet bytes. Nonetheless, considering the delay between frames could only be tested down to 118 ms, there still could exist other factors which could cause loss of bytes on received packet.

With tests of the modified I/O drivers for the SBC 544 completed, the frame and packet structured revisions took place. The UNID II was then retested using the frame and packet formats compatible with the CCITT X.25 recommendation. The test results were identical to the previous tests.

The data link software was not ready for operational testing and not tested. The SBC 88/45 software requires the addition of pointers and semaphores needed to communicate with the six tables located in system memory, declare statements modified to locate variables and tables at

the same memory location found in common with the SBC 544, and, finally, the hardware initialization routines necessary for operation of all three ports on the SBC 88/45.

Phase Four Testing

Phase four testing consisted of only an examination of the SBC 544 in the LSI-11 network. The original SBC 544 embedded software was configured in the network (see Figure 6-6 below) and data messages were then sent through the network using the software TEST3 at node D, GATEKE at node K, and GATEUE at node A.

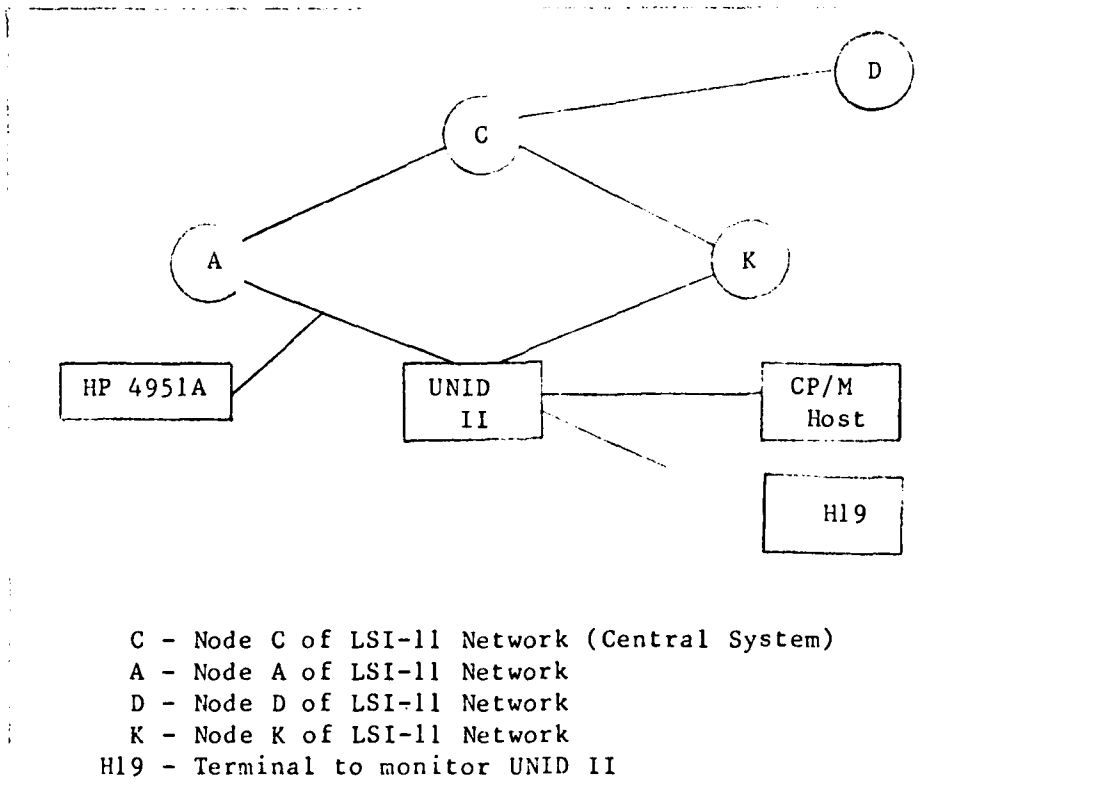


Figure 6-6. UNID II and NETOS Connection (15:5-14)

The UNID was inserted between node A and node K of the LSI-11 network. Channel 3 of the SBC 544 was hooked to node A and Channel 2 of the SBC

544 was hooked to node A. The routing message for the gateway software to send a message from node K to node A takes the form:

"Dxx3xxxxxxxx . . . xxxxxxxxxxx"

where "D" indicates node D is the source, "3" indicates the destination and "x" are arbitrary numbers for the remaining message bytes. The routing message placed the appropriate source and destination bytes in the datagram sent to the UNID to route the message to node A.

As described in (15:Chapter 5) the first datagram was received without error, but the second datagram was essentially "trashed" by the network. The modified software for the SBC 544 did correctly receive and pass the messages sent from node K to node A and the looping messages sent from node A to node K. The protocol analyzer was used to confirm proper handshaking between the UNID and the LSI-11 nodes and as an additional validation tool for the accurate transmission of the message.

Phase four testing was not accomplished on the 88/45 board software was not fully developed for the target EPROMs. The following paragraph describes the steps that would occur in Phase Four Testing.

The UNID II configuration consists of three boards in the six slot card cage. Slot J1 has the SBC 86/12A. Slot J3 has the SBC 88/45. Slot J5 has the SBC 544. The operational software on the SBC 544 must have all loop back procedures, dummy calls, and dummy responses associated with the SBC 88/45 removed. The loops on the SBC 88/45 remain unaltered for the initial Phase Four testing.

The host software developed for the SBC 544 will send datagrams to the SBC 544 network layer software. Successful exchanges between the

two SBCs for packets destined for the same port sending the message will result in messages received at the host terminal when the destination address is the host terminal. H19 terminals may be attached to the remaining ports of the SBC 544 to validate proper returns to the remaining local host channels. The SBC 86/12A will serve as a debugging tool for messages not successfully transversing the system memory boundary between the SBC 544 and SBC 88/45. For Phase Four testing, the SBC 88/45 transmit tables should be located in memory common between the SBC 86/12A and SBC 88/45. This will allow the SBC 86/12A monitor to access the frames residing in the SBC 88/45 transmit tables.

After validation of data exchanges between the SBC 544 and the SBC 88/45, the software loops within the data link software should be removed. The operational transmit and receive routines for the data link software will then be called to invoke the RS-422 physical layer interface. A null modem on channel B's RS-422 interface will be required to allow the the two channels to connect together for testing. The host program will then send messages to the then fully operational UNID II. Once the hardware interface has been validated as fully operational, the second UNID II software may be configured for the first test of an operational DELNET ring of two UNIDs.

#### Conclusion

This chapter outlined the test philosophy and methodology used to validate and integrate UNID II software. The test philosophy established terminology and validation requirements. The chapter then covered modifications to existing software testing tools. Next, the chapter briefly outlined the test stages developed for the validation

process. The final four sections of the chapter detailed Phase One module testing, Phase Two simulation testing, Phase Three operational testing, and Phase Four integration testing. Phase One and Phase Two tests were both "successfully completed." Phase Three testing of the SBC 544 was also completed. The Phase Three testing of the SBC 88/45 was not completed. This then precluded Phase Four testing of the SBC 88/45. The SBC 544, however, was successfully tested in the LSI-11 network. The next and final chapter of this thesis effort discusses the conclusions and recommendations of this thesis effort.



## Chapter VII

### Conclusions and Recommendations

#### Introduction

This Thesis effort sought to further the implementation of the CCITT X.25 protocol in the UNID II. While each of the tasks listed in Chapter I were addressed, not all were completed. Table 7-1 lists the tasks accomplished.

Table 7-1. Tasks Accomplished

1. The timing problem associated with bytes lost in received datagrams was corrected and validated on the LSI-11 network.
2. The software necessary to operate the UNID II off an Intel 210 and 230 development system was developed.
3. The frame and packet header formats were altered for consistency with the X.25 recommendation.
4. LAP B procedures were developed to the point that they may be integrated into operational SBC 88/45 software.
5. The "living document" approach used in this Thesis effort to provide sufficient and detailed information for following efforts provided in additional documentation on historical UNID II development and detailed procedures used in software development, design, integration, and validation.

### Conclusions

The UNID II design now includes fully operational TCP/IP and datalink software. While functions in both layers may be considerably expanded for a broader implementation, they do not now represent the main focus of UNID II development. The functional simulations developed by previous work (15) and considerably refined in this effort provides validated modules for an operational UNID II. The SBC 544 software will now support datagram transfer between local host channels. The SBC 544 allows up to four separate hosts to access any other hosts on the SBC 544. The access to the SBC 544 requires the use of the TCP/IP protocol, and the transmit request / transmit acknowledgement handshake.

The UNID II now has operational host test software on two separate systems (Intel 210, Intel 230) and under two separate operating systems (CP/M, ISIS). The modular design of the host software restricts modifications to the I/O module linked with the main program module. The modules developed during this thesis effort support the 8251 and 8251A USARTs commonly used in serial communications. Appendix D of this thesis discusses the details necessary to transport the source code to other devices. Appendix K of this thesis gives the three separate I/O modules for the host software and demonstrates the simplicity in writing the I/O drives in PL/M as opposed to an assembly language. Transportability of the host software was a key consideration in the selection of the language used and the modular refinement of the software. As such, the host software may easily be transported to the such systems as the Intel 310 and Intel 330 under either RMX or Xenix operating systems.

The packet layer for the UNID II now has the correct format for X.25 compatibility, though the packet layer functions are not fully implemented. The datalink layer for the UNID II now has the correct header format and, to the extent implemented, performs correctly. While a broader implementation may improve networking capacity and delay characteristics, such development is not necessary for a functional UNID II. The physical layer was addressed only in the design phase and at this point in the development cycle represents one of the next areas of development.

The major objective yet to be accomplished in the UNID II is embedding operational software on the SBC 88/45 and the integration testing, refinement, and validation of the SBC 544 and SBC 88/45.

#### Recommendation

As stated in previous UNID/DELNET research and development, this author recommends development continue on UNID II. The DELNET and UNID environment provides the developer a rich arena in networking related activities. The scope of knowledge gained from the DELNET/UNID environment is rich with the same activities associated with the OpenNet, ARPANET, DDNS, and other mainline networks. The software environment of UNID spans the entire development cycle, hence, UNID development details specifications, requirements, design, detailed design, implementation, integration, and validation. The interpretations on the specifications and requirements have been delimited more clearly as the project nears completion. Design changes made from one developer to the next have advanced and refined each stage of development. Integration of previous software with the developers own design has provided the "real

world" exercise missing in much of the previous course work of an educational environment. Test and validation techniques, rarely touched upon outside of software engineering circles, have become the major issues in UNID II development and have become the milestones by which the developer is judged. Furthermore, each of these stages of development forces the use of advanced design techniques (SADTs, Structure Charts, etc.).

At the other end of the development spectrum is the hardware related activities. The UNID II developer must have detailed understanding of interrupt programs, hardware interfaces such as X.21, RS-232-C, and RS-422. The details of converting the 'soft' code of software programs to machine readable code on EPROMs cannot be overlooked. At present, no less than four separate code conversions must currently be used before a single machine instruction can be executed on one of the SBCs.

The developer/officer directly benefits through the commonality of the UNID/DELNET design with current nodal processors and gateway switches going into the Defense Data Network (DDN) (110). The general strategy of these processes reflect the same approach taken with UNID design: common, off the shelf, SBCs, a multibus type card cage executing software designed using the ISO reference model. The UNID II design gives the officer/developer hands on experience with technology coming into the DoD inventory at this very time.

Specific recommendations are made in the following paragraphs.

1. Further development in the UNID II design should proceed with completion of the physical layer hardware initialization on the SBC

88/45. Seven ICs on the SBC 88/45 require programming. These devices will provide two RS-422 channels and one RS-232-C. The RS-232-C channel will be used to monitor operations within the SBC 88/45. One of the two RS-422 channels will require a null modem. A null modem simply interchanges the transmit and receive signals to allow interconnection of the two channels. The null modem interface provides orderly addition of other UNIDs to the DELNET ring.

2. Once the RS-422 interface is fully operational, the UNID II should be characterized to benchmark performance levels. Noted shortcomings would then guide further refinements to the UNID II software.

3. As now developed, further refinements to the UNID II software would come as top down extensions of the network and datalink layers. The network layer should have a fully functional RS-232-C interface. The interface should use RS-232-C circuits BA, BB, CA, CB, CC, and CF. The present software handshake sequence should then be removed.

4. For full datagram service, the minimal packet layer currently implemented requires, expansion. Once packet layer software becomes available, incorporation of virtual circuit service on the UNID becomes a simple process of extending packet layer services.

5. The datalink software should have the current one frame window extended to three or four frames. Intelligence should be 'built-in' the datalink software to handle simple error conditions without resetting the link.

6. Development of the UNID (0) gateway switch between DELNET rings should be addressed. The UNID (0) acts as a true gateway switch between UNID rings within DELNET. The hardware configuration of UNID (0) is

simply two SBC 88/45 boards replacing the SBC 544 and SBC 88/45 board pair. The datalink and packet layer software programs developed for an operational UNID II would require only minor changes for UNID (0). Features should be added to use one of the RS-232-C ports as a network monitor.

7. Further development should seriously look at the use of the 'C' programming language. While PL/M proves ideal for use in interrupt routines, the PL/M implementation can not achieve the same levels of abstraction as can an implementation in 'C'. Moreover, the available I/O routines in "C" greatly simplify the developers ability to extract information from the test program.

8. It is the contention of this author that the next development of the UNID II not proceed without the use of an advanced software development aid such as the RMX operating system. RMX provides the user a finite nucleus which may be embedded on the target EPROM of a SBC. The nucleus can be upgraded through additional operating system layers. Each layer tailors the RMX OS for the specific needs of the target SBC. The RMX facilities may range from a simple monitor type program to a fully functional macro computer with compilers, editors, dynamic debuggers, static debuggers, and a 'human interface'. No software need be written to add any of these features, no further hardware need be obtained. this author recommends use of the presently available SBC 86/30 board (128 k bytes on board RAM) as the target for the RMS nucleus. The SBC 86/30 would then replace the SBC 86/12A presently in use. Once the SBC 86/30 was in place, the SBC 544 and SBC 88/45 could be restrapped to use the system memory as location for their respective

programs. Then the software simply needs to be loaded into system memory through the SBC 86/30's RS-232-C port.

9. A simpler recommendation to recommendation 8 would be to mount the presently available 286 monitor firmware on the SBC 86/30. The SBC 286/10 monitor is partially functional on the SBC 86/30 and the SBC 86/30, unlike the SBC 286/10 has on board dual port RAM. The SBC 286/10 monitor has routines to up load and down load software from the Intel 230. In this fashion, the burning of EPROMs for each program change to operational UNID II software could be eliminated.

#### Concluding Remarks

The UNID/DELNET project has provided the basis for many thesis topics over the years. While the original palace of DELNET has been superseded by various local area networks, the UNID/DELNET project remains an invaluable educational resource. There is simply no replacement for the 'hands on experience' provided by the UNID/DELNET project.

## Bibliography

1. 1842 EEG/EEIC. An Engineering Assessment Toward Economic, Feasible and Responsive Base Level Communications Through the 1980's. Technical Report TR 78-5. Richards-Gebaur AFB, Missouri. October 1977.
2. Air Force Automated Systems Program Office (AFASPO) (AFCC). Private conversations with the program management staff regarding multi-user and multi-level security issues related to the Integrated-Service/Agency Automated Message Processing Exchange (I-S/A AMPE). Gunter AFS, AL, June-July 1982 and March-April 1983.
3. Andreoni, Gaetano, Le Moli, Gesualdo and Palazzo, Sergio "Sublayering in Standard Network Architectures," Computer Communications, 7: 17-22 (February 1984).
4. Arthurs, Edward, Gregory L. Chesson and Barton W. Stuck. "Theoretical Performance Analysis of Sliding Window Flow Control," IEEE Journal on Selected Areas in Communications, SAC-1: 947-959 (November 1983).
5. Baker, L. "USAF Prototype and Software Development for Universal Network Interface Device." MS thesis, AFIT-GCS-EE-80D-4. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1980 (AD A100787).
6. Bartoli, Paul D. "The Application Layer of the Reference Model of Open Systems Interconnection," Proceedings of the IEEE, 71: 1404-1407 (December 1983).
7. Bertine, H. V. "Physical Level Interface and Protocols," Computer Networks: A Tutorial, IEEE Computer Society: 97 - 124 (1984).
8. Blumberg, Robert, "Access Protocols for X.25 Local Area Network," IEEE Computer Society International Conference/Fall: 13 - 19 (1983).
9. Borgsmiller, Michael. "The Serial Communications Interface Board." Project Report, School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, March 1983.
10. Brown, E. "USAF Prototype Universal Network Interface Device." MS thesis, AFIT-GE-EE-79-8. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1979 (AD A080173).
11. Burg, Fred M. "Design Considerations for using the X.25 Packet Layer in Data Terminal Equipment," INFOCOM: 180 - 188 (April 1984).



12. Bytek Corporation. EPROM Programmer Manual. Manufacture's data. Santa Clara CA, 1982.
13. Callon, Ross "Internetwork Protocol," Proceedings of the IEEE, 71: 1388-1393 (December 1983).
14. Carlson, D. E. "Bit - Orientated Data Link Control Procedures ," Computer Networks: A Tutorial, IEEE Computer Society: 125 - 138 (1984).
15. Childress, C. T. "Continued Development and Implementation of the Universal Network Interface Device (UNID) II in the Digital Engineering Laboratory Network (DELNET)". MS Thesis, AFIT-GE-EE-84D-17. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB, OH, December 1984.
16. Childress, C. T. Telephone Conversations regarding UNID II design, May - September 1985.
17. Cole, Kenneth. Private conversations regarding data structures. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, June - September 1984.
18. Conard, James W. "Services and Protocols of the Data Link Layer," Proceedings of the IEEE, 71: 1378-1383 (December 1983).
19. Cuomo, Gennaro. "Continued Development of the Universal Network Interface Device." MS thesis, AFIT-GE-EE-82D-28. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1982.
20. Deital, Harvey M., An Introduction to Operationing Systems, Reading, Massachussetts, Addison-Wesly, 1984.
21. DOD Standard: Transmission Control Protocol Specification. Arlington VA: Defense Advanced Research Projects Agency (DARPA), September 1981.
22. DOD Standard: Internet Protocol Specification. Arlington VA: Defense Advanced Research Projects Agency (DARPA), September 1981.
23. Dolpine, Richard, Principles of Data Communications, Carnegie Press, Madison, NJ, 1984.
24. EIA Standard RS-232C. Interface Between Data Terminal Equipment and Data Communication Equipment Employing Serial Binary Data Interchange. Washington, D.C.: Electronic Industrial Association, April 1975.

25. EIA Standard RS-449. General Purpose 37 Position Interface for Data Terminal Equipment and Data Circuit Terminating Equipment Employing Serial Binary Data Interchange. Washington, D.C.: Electronic Industrial Association, November 1977.
26. Emmons, Willard F. and Chandler, A.S. "OSI Session Layer: Services and Protocols," Proceedings of the IEEE, 71: 1397-1400 (December 1983).
27. Fairchild Camera and Instrument Corporation. Microprocessor Products Data Book. Manufacturer's data. Santa Clara, California: Fairchild Camera and Instrument Corporation, January 1983.
28. Folts, Harold C., Harry R. Karp, Data Communications Standards, McGraw-Hill, New York, NY, 1979.
29. Freeman, Peter. "Fundamentals of Design," Tutorial on Software Design Techniques, IEEE Computer Society: 2 - 22 (1984).
30. Geist, John W. "Development of the Digital Engineering Laboratory Computer Network: Host-to-Node/Host-to-Host Protocols." MS thesis, AFIT-GCS-EE-81D-8. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1981.
31. Gravin, Andrew G. "Preliminary Design of a Computer Communications Network Interface Device Using INTEL 8086 and 8089 16-Bit Microprocessors." MS thesis, AFIT-GCS-EE-81D-9. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1981.
32. Hartrum, Thomas C. Private conversations regarding the AFIT LSI-11 Network Operating System (NETOS). School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, June - September 1985.
33. Hazelton, Craig H. "Continued Development and Implementation of the Protocols for the Digital Engineering Laboratory Network." MS thesis, AFIT-GE-EE-82D-37. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1982.
34. Hewlett-Packard Company. HP 4951A protocol analyzer Operating Manual No. 04951-90012, November 1984. HP Company/Colorado Telecommunications Division.

35. Hobart, William C., Jr. "Design of a Local Computer Network for the Air Force Institute of Technology Digital Engineering Laboratory." MS thesis, AFIT-CE-EE-81M-3. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, March 1981.
36. Hollis, Lloyd L. "OSI Presentation Layer Activities" Proceedings of the IEEE, 71: 1401-1403 (December 1983).
37. Hunt, Ray "Open Systems interconnection - the transport layer protocol," Computer Communications, 7: 186-197 (August 1984).
38. Specification for Message Format for Computer Based Message Systems. Proposed Federal Information Processing Standard. Washington DC: Institute for Computer Sciences and Technology, National Bureau of Standards, Department of Commerce, April 1982.
39. Specification of a Transport Protocol for Computer Communications, Volume 1: Overview and Services. Draft Report ICST/HLNP-83-1. Washington DC: Institute for Computer Sciences and Technology, National Bureau of Standards, Department of Commerce, January 1983.
40. Specification of a Transport Protocol for Computer Communications, Volume 2: Class 2 Protocol. Draft Report ICST/HLNP-83-2. Washington DC: Institute for Computer Sciences and Technology, National Bureau of Standards, Department of Commerce, February 1983.
41. Specification of a Transport Protocol for Computer Communications, Volume 3: Class 4 Protocol. Draft Report ICST/HLNP-83-3. Washington DC: Institute for Computer Sciences and Technology, National Bureau of Standards, Department of Commerce, February 1983.
42. Specification of a Transport Protocol for Computer Communications, Volume 4: Service Specifications. Draft Report ICST/HLNP-83-4. Washington DC: Institute for Computer Sciences and Technology, National Bureau of Standards, Department of Commerce, January 1983.
43. Specification of a Transport Protocol for Computer Communications, Volume 5: Guidance for the Implementor. Draft Report ICST/HLNP-83-5. Washington DC: Institute for Computer Sciences and Technology, National Bureau of Standards, Department of Commerce, January 1983.

44. Specification of a Transport Protocol for Computer Communications, Volume 6: Guidance for Implementation Selection. Draft Report ICST/HLNP-83-6. Washington DC: Institute for Computer Sciences and Technology, National Bureau of Standards, Department of Commerce, February 1983.
45. Intel Corp. AEDIT-86 Text Editor User's Guide. Manufacturer's data, 121956-001. Intel Corp., Santa Clara CA, 1982.
46. Intel Corp. "Communications," ;Comments, Intel Corp., Santa Clara CA: Chapter 6 (June 1985).
47. Intel Corp. Development Systems Handbook. Manufacturer's data, 210940-001. Intel Corp., Santa Clara CA, May 1983.
48. Intel Corp. Distributed Control Modules Databook. Manufacturer's data, 230972-001. Intel Corp., Santa Clara CA, 1984.
49. Intel Corp. iAPX 86, 88 Family Utilities User's Guide. Manufacturer's data, 121616-003. Intel Corp., Santa Clara CA, 1982.
50. Intel Corp. iAPX 86/88, 186/188 User's Manual: Programmer's Reference. Manufacturer's data, 210911-001. Intel Corp., Santa Clara CA, May 1983.
51. Intel Corp. ICE-85B In-Circuit Emulator Operating Instructions for ISIS-II Users. Manufacturer's data, 980463-003. Intel Corp., Santa Clara CA, 1982.
52. Intel Corp. ICE-86A/ICE-88A Microsystems In-Circuit Emulator Operating Instructions for ISIS-II Users. Manufacturer's data, 162554-002. Intel Corp., Santa Clara CA, 1982.
53. Intel Corp. Intellec Series III Microcomputer Development System Programmer's Reference Manual. Manufacturer's data, 121618-003. Intel Corp., Santa Clara CA, 1981.
54. Intel Corp. Intellec Series III Microcomputer Development System Product Overview. Manufacturer's data, 121575-002, 1981. Intel Corp., Santa Clara CA, 1981.
55. Intel Corp. Intellec Series III Microcomputer Development System Console Operating Instructions. Manufacturer's data, 121609-003. Intel Corp., Santa Clara CA, 1981.
56. Intel Corp. iSBC 86/12A Single Board Computer Hardware Reference Manual. Manufacturer's data, 983074-01. Intel Corp., Santa Clara CA, undated.

57. Intel Corp. iSBC 88/45 Advanced Data Communications Processor Board Hardware Reference Manual. Manufacturer's data, 143824-001. Intel Corp., Santa Clara CA, 1983.
58. Intel Corp. iSBC 544 Intellegent Communications Controller Board Hardware Reference Manual. Manufacturer's data, 980616B. Intel Corp., Santa Clara CA, 1983.
59. Intel Corp. iSBC 957 Intellec iSBC 86/12A Interface and Executive Package User's Guide. Manufacturer's data, 9800743A, Intel Corp., Santa Clara CA , 1978.
60. Intel Corp. ISIS-II User's Guide. Manufacturer's data, 980306-05. Intel Corp., Santa Clara CA, 1979.
61. Intel Corp. ISIS-II PL/M-80 Compiler Operator's Manual. Manufacturer's data, 980300-004. Intel Corp., Santa Clara CA, undated.
62. Intel Corp. ISIS-II PL/M-86 Compiler Operator's Manual. Manufacturer's data, 980478-004. Intel Corp., Santa Clara CA, undated.
63. Intel Corp. MCS-86 Absolute Object File Formats. Manufacturer's data, 980821A. Intel Corp., Santa Clara CA, 1977.
64. Intel Corp. MCS-86 Software Development Utilities Operating Instructions for ISIS-II Users. Manufacturer's data, 980639B. Intel Corp., Santa Clara CA, undated.
65. Intel Corp. Memory Components Handbook. Manufacturer's data, 210830-002. Intel Corp., Santa Clara CA, 1983.
66. Intel Corp. Microprocessor and Peripheral Handbook. Manufacturer's data, 210844-001. Intel Corp., Santa Clara CA, 1983.
67. Intel Corp. OEM Systems Handbook. Manufacturer's data, 210941-004. Intel Corp., Santa Clara CA, 1984.
68. Intel Corp. OpenNet Product Guide, Intel Corp., Santa Clara, CA, 1985,
69. Intel Corp. Peripheral Design Handbook, Intel Corp., Santa Clara CA, August 1981.
70. Intel Corp. PL/M-80 Programming Manual. Manufacturer's data, 980268-002. Intel Corp., Santa Clara CA, undated.
71. Intel Corp. PL/M-86 Programming Manual. Manufacturer's data, 980466-003. Intel Corp., Santa Clara CA, undated.

72. Intel Corp. Prototyping with the 8089 I/O Processor. Manufacturer's data, AP-89. Intel Corp., Santa Clara CA, May 1980.
73. Intel Corp. "Serial Interface Connection Chart," Comments Intel Corp., Santa Clara, CA: Chapter 1 (June 1985).
74. Intel Corp. Software Handbook. Manufacturer's data, 230766-001. Intel Corp., Santa Clara CA, 1984.
75. Interphase Corp. LNC 5180 Local Area Network Controller Users's Guide. Manufacturer's data, Interphase, Dallas, TX, September 1983.
76. Kernighan, Brian W., Dennis M. Ritchie, The C Programming Language, Prentice-Hall Inc., Englewood Cliffs, NJ, 1978.
77. Knightson, Keith G. "The Transport Layer Standardization," Proceedings of the IEEE, 71: 1394-1396 (December 1983).
78. Kochan, Stephen G., Programming in C, Hayden Book Company, Hasbrock Heights, NJ, 1983.
79. Limb, John O. "Performance of Local Area Networks at High Speed," IEEE Communication Magazine, 22: 41-45 (August 1984).
80. Lin, Shu, and Daniel J. Costello, Jr. Error Control Coding: Fundamentals and Applications. Englewood Cliffs, New Jersey: Prentice-Hall Inc., 1983.
81. McClelland, Frank M. "Services and Protocols of the Physical Layer," Proceedings of the IEEE, 71: 1372-1377 (December 1983).
82. Madnick, Stuart E. and John J. Donovan. Operating Systems. New York, New York: McGraw-Hill Book Co., 1974.
83. Magee, F. R., Jr., R. P. Devey. "An Internal Packet Network Protocol and Buffer Management Scheme for an X.25 Based Network," INFOCOM: 481 - 484 (April 1983).
84. Matheson, William F. "Continued Development of a Universal Network Interface Device Using the INTEL 8086 and 8089 16-Bit Microprocessors." MS thesis, AFIT-GE-EE-83D-42. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1983.

85. Merwin, Richard E. Secure Operating System Technology Papers for the Seminar on the DoD Computer Security Initiative Program. American Federation of Information Processing Societies Conference Proceedings, National Computer Conference. Montvale NJ: American Federation of Information Processing Societies, Inc., June, July 1979.
86. Military Standard. File Transfer Protocol. MIL-STD-1780 (Draft), US Government Printing Office, Arlington, Virginia: 1983.
87. Military Standard. Internet Protocol. MIL-STD-1777, US Government Printing Office, Arlington, Virginia: 1983.
88. Military Standard. Simple Mail Transfer Protocol. MIL-STD-1781 (Draft), US Government Printing Office, Arlington, Virginia: 1983.
89. Military Standard. Transmission Control Protocol. MIL-STD-1778, US Government Printing Office, Arlington, Virginia: 1983.
90. Military Standard. TELNET Protocol. MIL-STD-1782 (Draft), US Government Printing Office, Arlington, Virginia: 1984.
91. Palmer, Donald E. "Design of a Prototype Universal Network Interface Device Using INTEL 8086 and 8089 16-Bit Microprocessors." MS thesis, AFIT-GCS-EE-82D-52. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1982.
92. Papp, Charles E. "Prototype DELNET Using the Universal Network Interface Device." MS thesis, AFIT-GE-EE-81D-46. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1981.
93. Phister, Paul W. Jr. "Protocol Standard and Implementation Within the Digital Engineering Laboratory Computer Network (DELNET) Using the Universal Network Interface Device (UNID)." MS thesis, AFIT-GE-EE-83D-58. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1983.
94. Pickens, R. Andrew. "Wideband Transmission Media II: Satellite Communications," Computer Communications, Volume 1, Principles. edited by Wushow Chou. Englewood Cliffs, N. J.: Prentice-Hall, Inc., 1983.
95. Pingry, Julie. "Local Area Networking Becomes A Standard Feature," Digital Design, 14: 70-83 (March 1984).
96. Ravenscroft, D. "Electrical Engineering Digital Design Laboratory Communications Network." MS thesis, AFIT-GCS-EE-78-16. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1978 (AD A064729).

97. Rivest, Ronald L., Adi Shamir and Len Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. Massachusetts Institute of Technology publication MIT/LCS/TM-82. Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge MA, April 1977.
98. Rudin, Harry "An Informal Overview of Formal Protocol Specification," IEEE Communication Magazine, 23: 46-52 (March 1985).
99. Rubin, Izhak and Luis F. M. De Moraes. "Message Delay Analysis for Polling and Token Multiple-Access Schemes for Local Communication Networks," IEEE Journal on Selected Areas in Communications, SAC-1: 935-946 (November 1983).
100. Rybczynski, A.M. "A Common X.25 Interface to Public Data Networks," Computer Networks, 4: 97-110 (1980).
101. Sauer, Charles H. and K. Mani Chandy. Computer Systems Performance Modeling. Englewood Cliffs, New Jersey: Prentice-Hall Inc., 1981.
102. Sauer, Charles H. and Edward A. MacNair. Simulation of Computer Communication Systems. Englewood Cliffs, New Jersey: Prentice-Hall Inc., 1983.
103. Schindler, Sigram, Luckenbach, Thomas and Steinacker, Michael "X.21 as a Universal Digital Service Access Interface," Computer Communications, 5: 298-306 (December 1982).
104. Sinkov, Abraham. Elementary Cryptanalysis. Washington DC: Mathematical Association of America, 1966.
105. Sirbu, Marvin A. "Standards Setting for Computer Communication: The Case of X.25," IEEE Communications Magazine, 23: 35-45 (March 1985).
106. Sloman, M.S. "Standards and Protocols," Computer Communications, 1: 310-328 (December 1978).
107. Sluzevich, Sam C. "Preliminary Design of a Universal Network Interface Device." MS thesis, AFIT-GE-EE-78-41. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, December 1978 (AD A064059).
108. Spear, Mark C. "Hardware Design and Implementation of the Universal Network Interface Device (UNID)." MS thesis, AFIT-GE-EE-84N-XX. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, March 1984.



109. Steinmetz, Jay S. "A Secure Computer Network." MS thesis, AFIT-GCS-EE-82D-34. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, November 1982.
110. Systems Development Corp., SDC Introduces the MIL/INT Network System, Manufacture's Data. System Development Corp., Communications System, Santa Monica, CA, 1984.
111. Transport Layer Specification. Draft specification. Documentation and programs available on UNIX compatible magnetic tape. Washington DC, National Bureau of Standards, December 1983.
112. Tannenbaum, Andrew S. Computer Networks. Englewood Cliffs, New Jersey: Prentice-Hall Inc., 1981.
113. Ware, Christine "The OSI Network Layer: Standards to Cope with the Real World," Proceedings of the IEEE, 71: 1384-1387 (December 1983).
114. Witt, Michael. "An Introduction to Layered Protocols." Byte: 385-398, September 1983.
115. The X.25 Protocol and Seven Other Key CCITT Recommendations: X.1, X.2, X.3, X.21, X.21(bis), X.28, and X.29. Lifetime Learning Publications, Belmont CA, 1981.
116. Yu, W., Majithia, J.C., and Wong, J.W. "Access Protocols for Circuit/Packet Switching Networks," Computer Communications, 4: 271-283 (1980).
117. Zilog, Inc. PLZ User Guide. Manufacturer's data, 03-3096-01. Zilog Inc., Cupertino CA, July 1979.

## Appendix A

### UNIB II Data Flow Diagrams

This appendix contains the Data Flow Diagrams (DFDs) for the UNIB II which were developed in a previous thesis effort (SI:26-14). These DFDs show the UNIB II message processing functions and the internal flow of messages between the local and network I/O hardware ports. The DFDs are still current and are shown in this thesis for completeness. The DFDs are presented in the following order:

Figure		Page
A-1.	UNIB II Overview	A-2
A-2.	Input Local Information	A-3
A-3.	Format according to Output Protocol	A-4
A-4.	Transmit Network Message	A-5
A-5.	Input Network Information	A-6
A-6.	Transmit Local Information	A-7

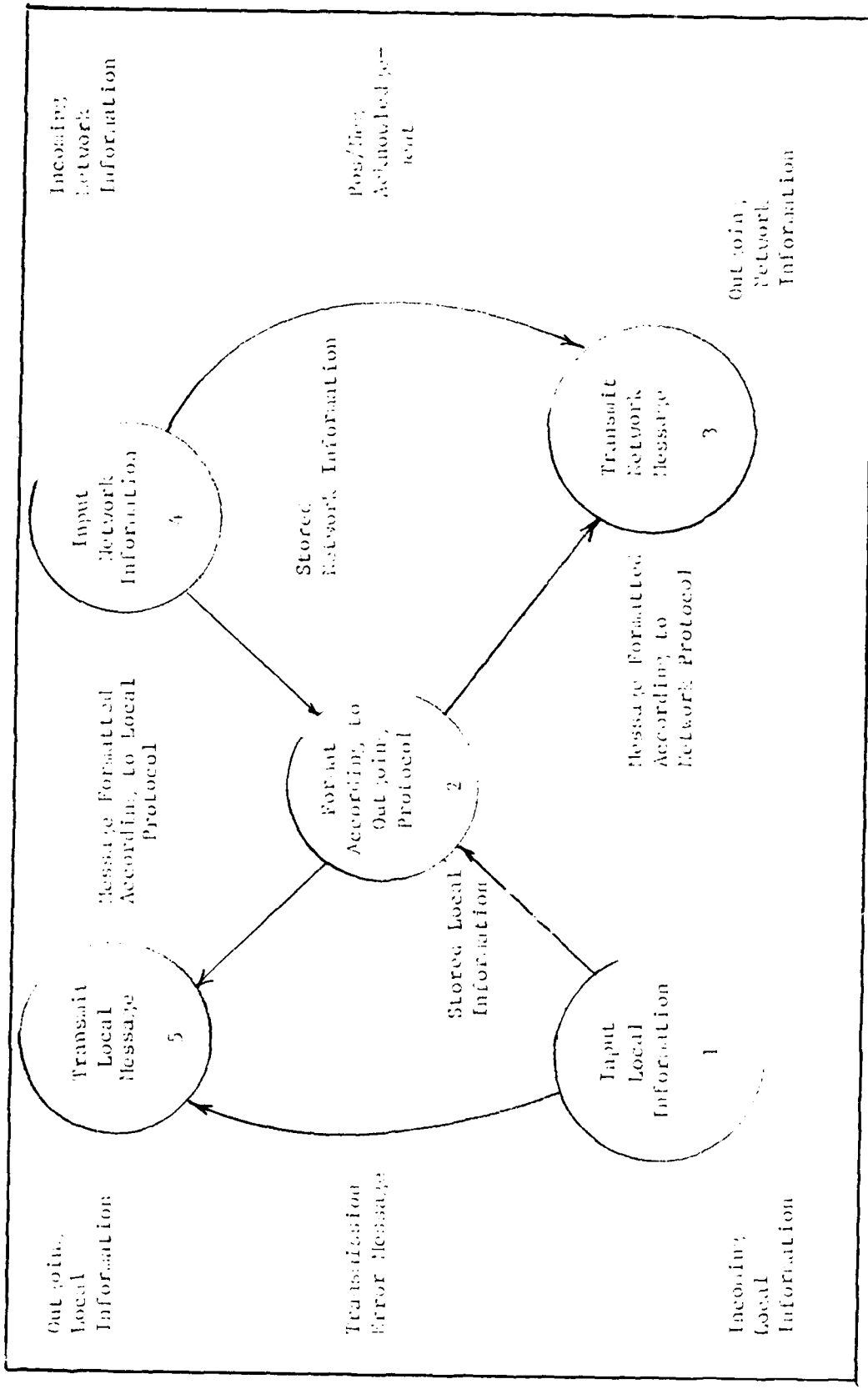


Figure A-1. UNID II Overview (31)

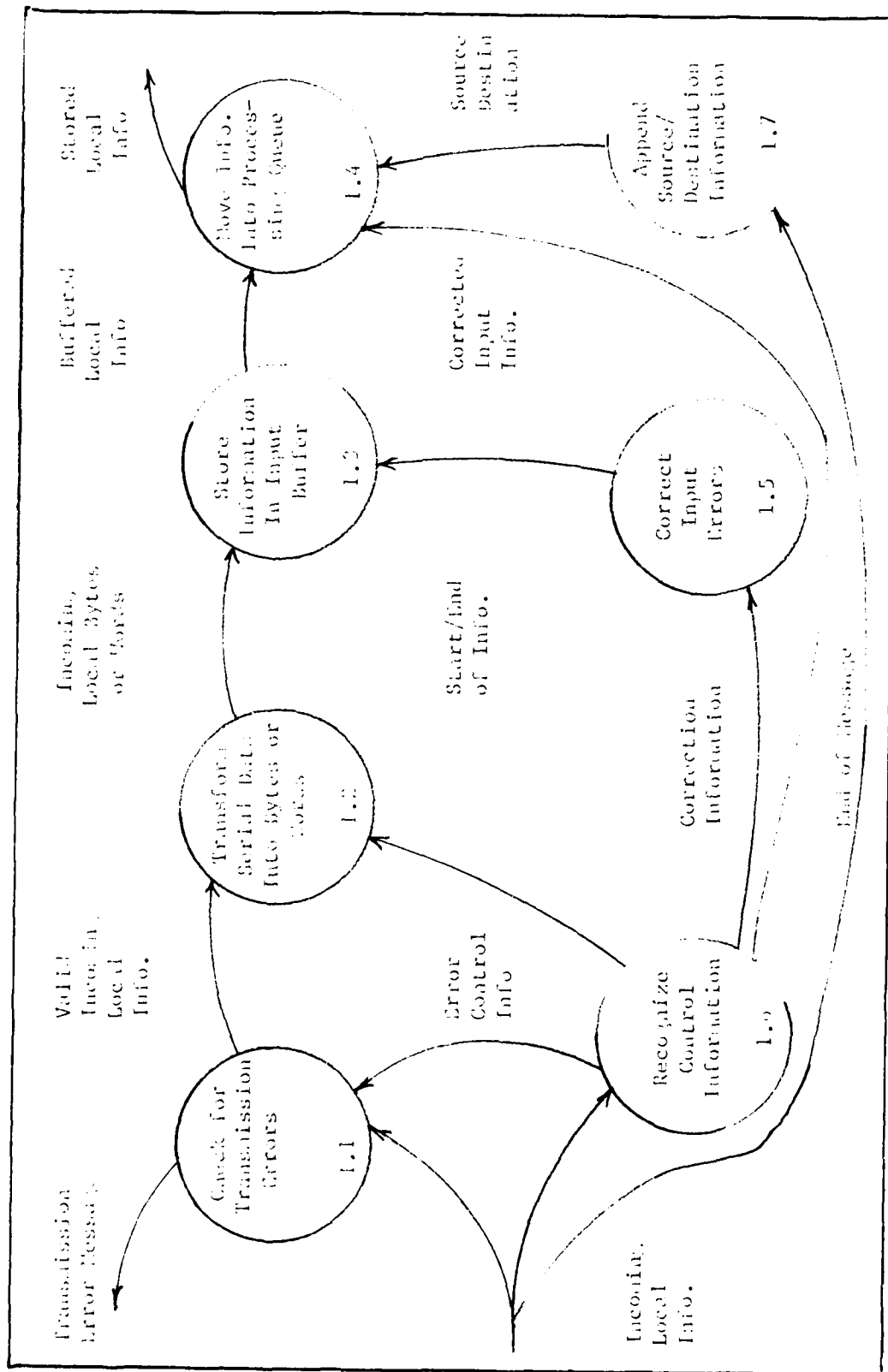


Figure A-7. Input Local Information (21)

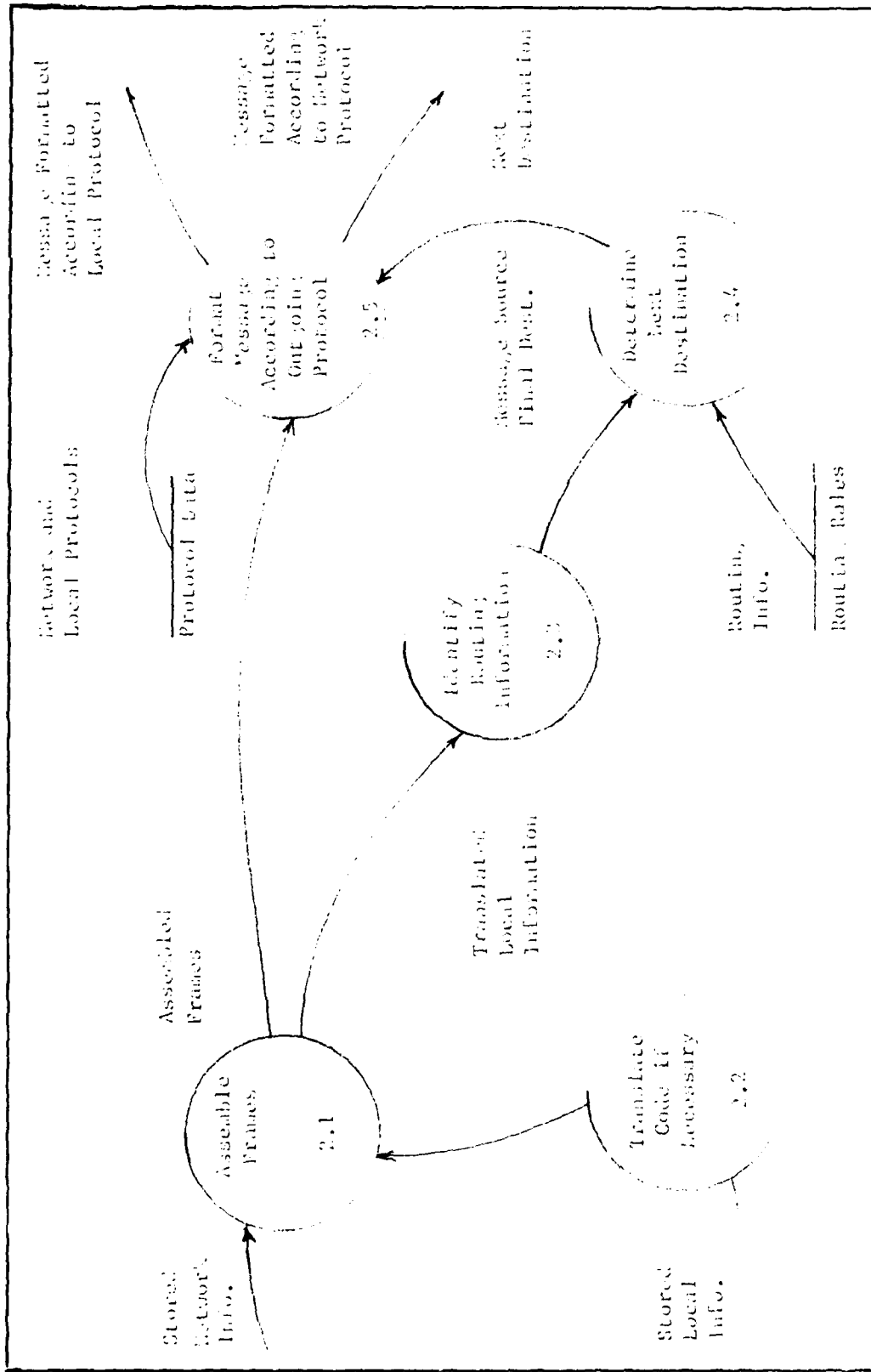


Figure A-3. Format according to Outgoing Protocol (31)

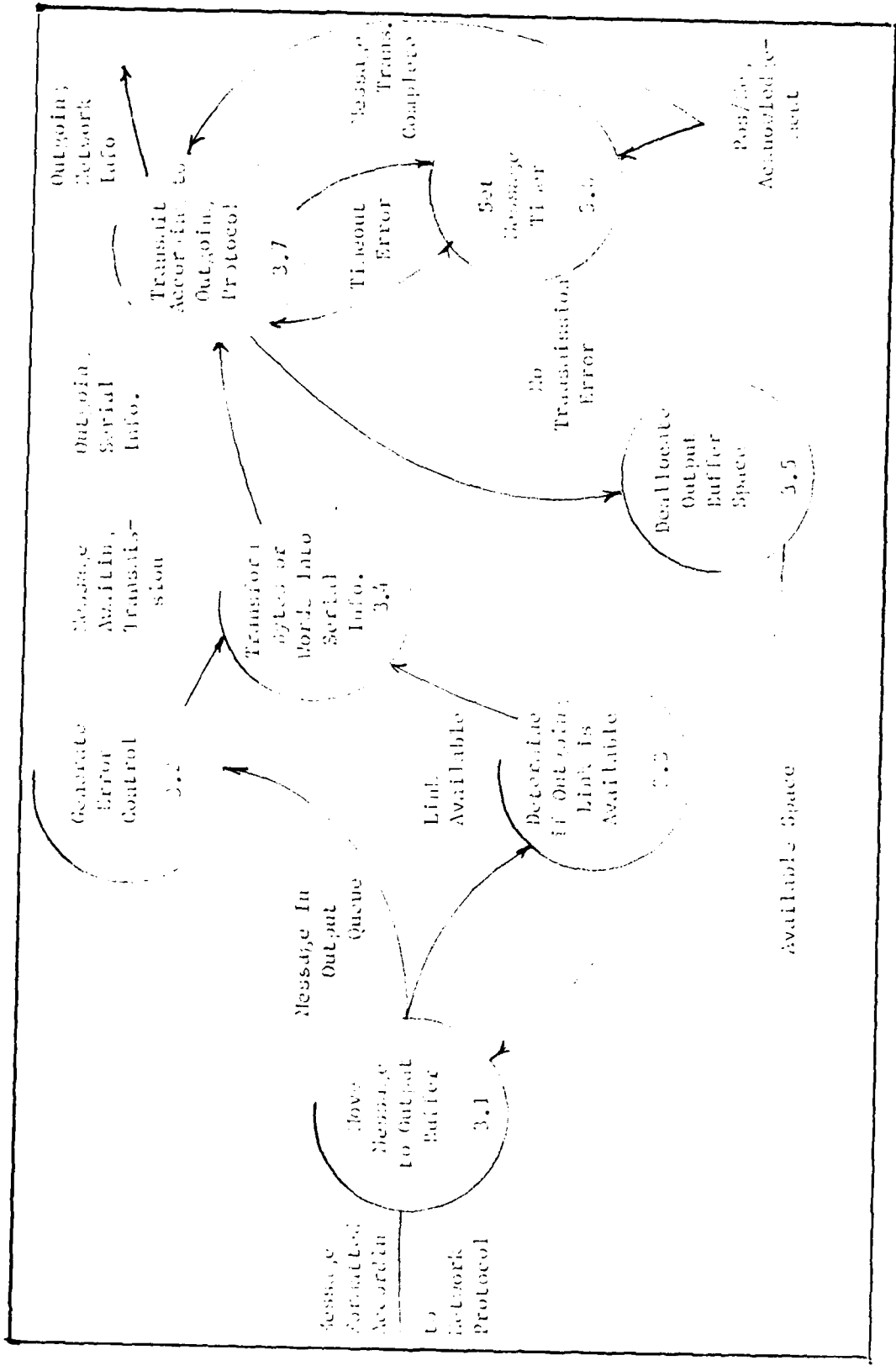


Figure A-4. Transmit Network Message (31)

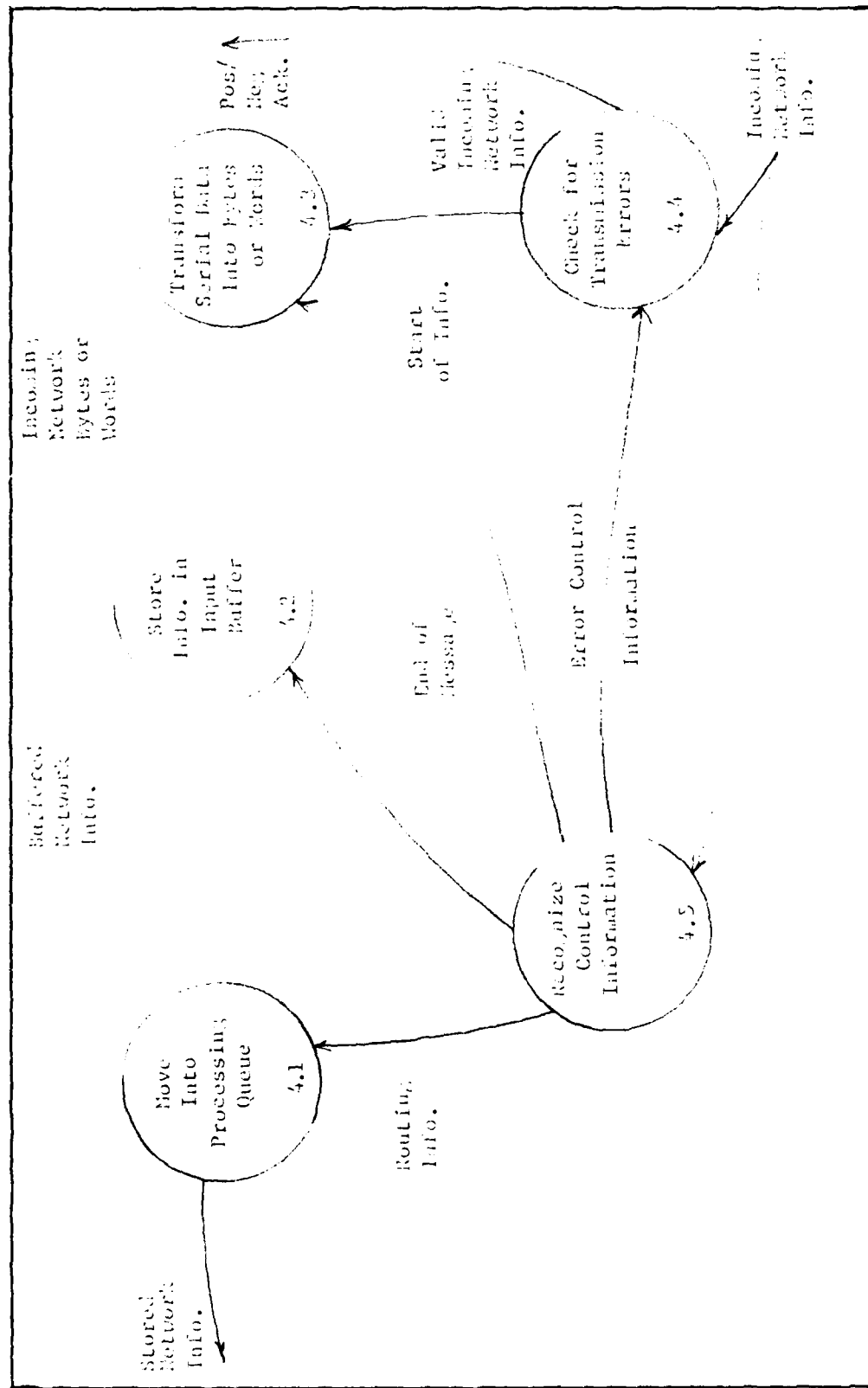


Figure A-5. Input Network Information (31)

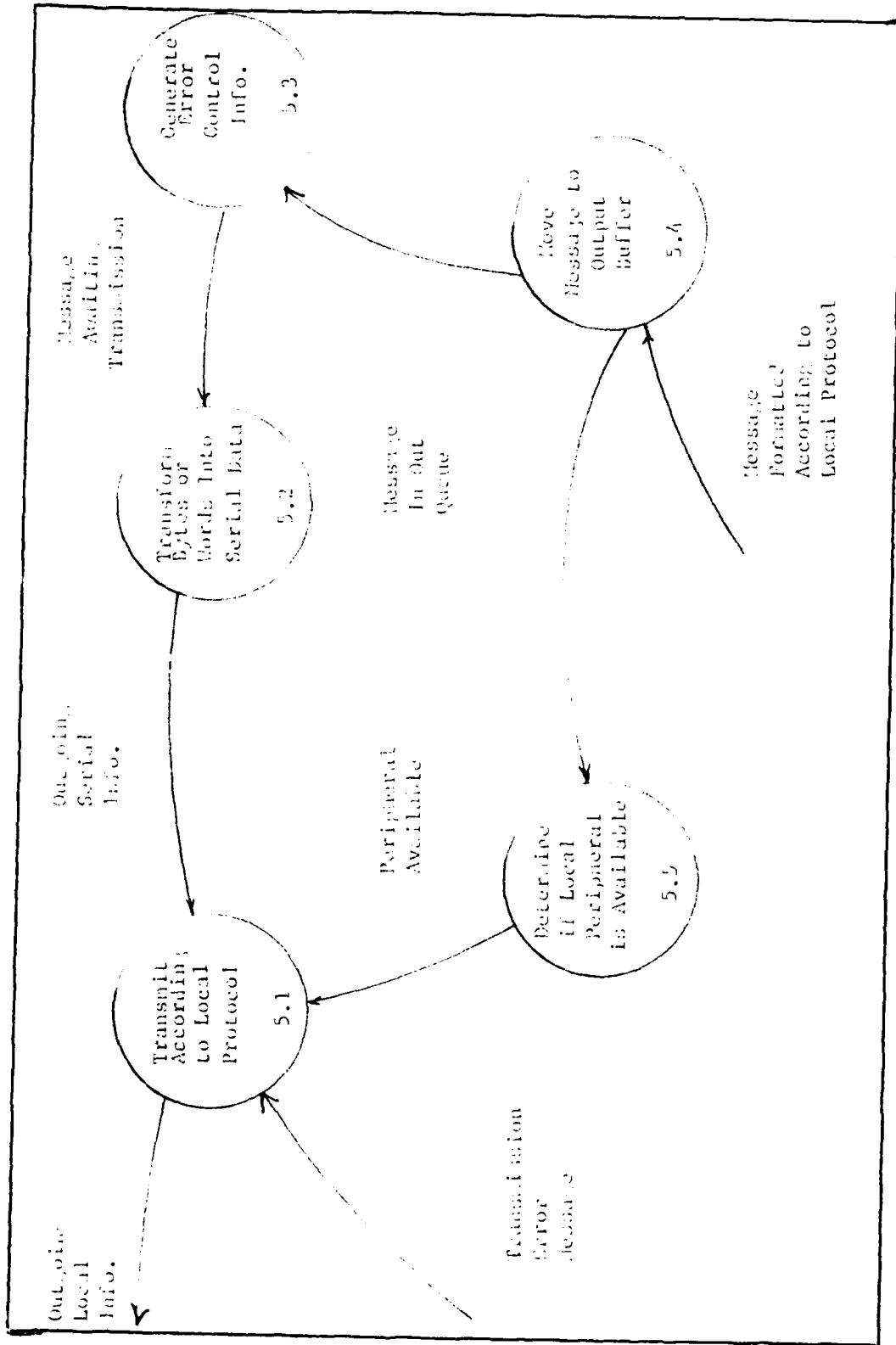


Figure A-6. Transmit Local Message (31)



Appendix B (15:Appendix B)

RS-232C and RS-422 Signals

The network layer (SBC 544 or similar hardware interface on the local host side of the UNID II) will use the RS-232C standard to interface with the host computers or other DELNET networks (93:Chapter 1, Chapter 2), such as the NETOS (32). The data link layer (SBC 83/45 or similar hardware interface on the network side of the UNID II) will use the RS-422 standard to interface one UNID with another UNID (67:Chapter 1, Chapter 2).

Each standard (24, 25) has a large number of signals specified not all of which are used by the UNID IIs (93: Appendix B). The following figures indicate the signals that are implemented in the UNID and DELNET. Signal direction into or out of the UNID II is shown for the RS-422 interface. Signal direction into or out of the UNID II is not shown for the RS-232C interface as the SBC 544 board is jumper configurable for either a DTE or a DCE connection. The default configuration for the RS-232C ports on the SBC 544 board is set to a DCE to allow host host computers and terminals to directly connect to the UNID II without the use of a null modem.

DB-25 Pin Number	Signal Nomenclature	Implemented in DELNET
1	Frame Ground	X
2	Transmit Data	X
3	Receive Data	X
4	Request to Send	X
5	Clear to Send	X
6	Data Set Ready	X
7	Signal Ground	X
8	Receive Line Signal Detect	(X)
9	Unassigned	
10	Unassigned	
11	Unassigned	
12	Secondary Receive Signal Detect	
13	Secondary Clear to Send	
14	Secondary Transmit Data	
15	Transmit Signal Element Timing	
16	Secondary Receive Data	
17	Receive Signal Element Timing	
18	Unassigned	
19	Secondary Request to Send	
20	Data Terminal Ready	X
21	Signal Quality Indicator	
22	Ring Detector	(X)
23	Data Signal Rate Select (DTE)	
24	Data Signal Rate Select (DCE)	
25	Unassigned	

NOTE: Pins 8 and 22, marked (X), are not required when the UNID is hard wired to a host computer or terminal. These pins would normally be used when a particular port is connected to a modem.

Figure E-1. RS-232C Pin Assignments

The local side connected to host computers or terminals will use the RS-232C signals shown in Figure E-1. It is assumed that the RS-232C interface uses a standard DB-25 connector.

DB-37 Pin Number	Signal Nomenclature	Direction		Implemented in DELNET
		In	Out	
1	Shield			X
2	Signal Rate Indicator A			
3	Spare B			
4	Send Data A		X	X
5	Send Timing A		X	X
6	Receive Data A	X		X
7	Request to Send A		X	X
8	Receive Timing A	X		X
9	Clear to Send A	X		X
10	Local Loopback A			
11	Data Mode A			
12	Terminal Ready A		X	X
13	Receiver Ready A	X		X
14	Remote Loopback			
15	Incoming Call	X		
16	Select Frequency Signalling; Rate Indicator			
17	Terminal Timing		X	
18	Test Mode A			
19	Signal Ground			X
20	Receive Common			
21	Spare A			
22	Send Data B		X	X
23	Send Timing B		X	X
24	Receive Data B	X		X
25	Request to Send B		X	X
26	Receive Timing B	X		X
27	Clear to Send B	X		X
28	Terminal in Service A			
29	Data Mode B			
30	Terminal Ready B			X
31	Receiver Ready B			X
32	Select Standby A			
33	Signal Quality		X	
34	New Signal			
35	Terminal Timing		X	
36	Standby Indicator			
37	Send Common			

NOTE: The -A and -B suffixes on the signal nomenclature refer to the non-inverted and inverted outputs/inputs of the RS-422 signals. If the signals use a balanced signalling method and a reversed connection will invert the desired signal.

Figure B-2. RS-422 Pin Assignments

The UNIDs are connected using the RS-422 standard on the data link layer (subnetwork) side of the UNID. Figure B-2 shows the pin assignments and indicates which signals are currently implemented in the BELNET. In this figure, the direction into and out of the UNID is also shown to clarify the use of a null modem required on one of the two high speed network ports. It is assumed that the RS-422 interface uses a standard DB-37 connector.

## Appendix C

### Hardware Configuration for the UNID II

This appendix gives the strapping values necessary to operate the SBC 30/12A, SBC 344, SBC 30/45, AMCO card cage as part of the UNID II.

#### SBC 30/12A strapings

The following pins are interconnected either through wire wrapping or strapping bars:

7-8	23-25	66-75	92-93
9-10	30-31	83-70	103-104
11-20	32-33	93-77	105-106
13-14	24-35	76-78	127-128
15-16	31-32	71-79	129-130
17-18	54-55	72-80	143-144
19-20	56-57	73-81	151-152
24-25	59-60	67-68	

The DIP switches are set as follows (on = 0): 01010010 (pins 1 - 8).  
The above configuration is for a SBC 30/12A with the Intel SBC 344 memory expansion board. EPROMs are placed in their sockets according to the following:

- A23 Low Even memory
- A24 High Even memory
- A46 Low Odd memory
- A47 High Even memory

The jumpers for the SBC 344 board are installed as follows:

2-4	3-33	74-75
6-7	41-42 (add)	76-83
9-11	43-44 (remove)	79-71
13-14	45-46	82-83
17-18	47-48 (remove)	94-95
20-21	49-50 (remove)	96-99 (add)
23-25	55-54 (add)	100-101
26-27	55-56 (remove)	104-105
29-30	57-58	
22-23	72-73	

(add) - requires adding jumper bar on "as shipped board"

(remove) - requires removing jump bar on "as shipped board"

The lead numbers on connectors J1 - J4 don't correspond to the RS-232-C number convention; (55) gives the relationship between the J-1 - J-4 pinout number and the proper RS-232-C pin number.

As shipped, the SBC 344 is setup as a DTE. Implementation in the UNID has it setup as a DCE. This requires that the following leads to be crossed:

RD, RxD  
CTS, RTS  
LTR, DSR

The SBC 3445 board requires the following strapings:

1-2	41-43	114-115	157-158
3-4	72-43	117-118	140-144 (remove)
7-8	74-75	119-120 (add)	141-143
13-14	75-77	121-122 (remove)	142-146
19-22	82-93	123-124	155-157
21-24	83-94	125-126	157-165
26-29	84-95	132-133	159-165
32-33	85-96	150-151	171-172
36-37	107-108	156-158	173-176
38-39	110-111	152-155	175-178
41-39	112-113	135-136	

In general, the SBC 3445 is strapped correctly with the default settings.

## Appendix I

### HOST SYSTEM MODIFICATION PROCEDURES

#### INTRODUCTION

Modifying the UNID II host software requires three changes to be made. First, external calls to the existing operation system must be changed to call external procedures associated with the new operating system. Second, the I/O driver routines for the new serial I/O device must replace the previous I/O driver routines. Third, the software needs to be relocated to a point in system memory where execution is expected to take place. In replacing the BIOS calls in the assembly companion program (SBS.ASM in Childress's Thesis or HOSTL.ASM in this Thesis) the public references to coast, coin, count, hpos, and exit may all be removed from the program file. At this point the entire file of internal calls could be easily written in PL/M rather than assembly.

#### The Details

I/O driver routines implemented in Childress' SBS.ASM program support the MB 8250 universal asynchronous receiver/transmitter (UART). The I/O driver routines implemented in HOSTL.ASM support the 8251/8251A USART. Due to the commonality between the two devices, the only change required in the software was proper initialization of the USART. Since many computer systems use the 8251/8251A as the serial interface device, the only change necessary to adapt the ISO level routines to a new host would be in the I/O port addresses which are as follow:

SDATA - Serial data port (0F40)  
 SCONT - Serial control port (0F50)  
 SSTAT - Serial status port (0F60)  
 COUNTER - Mode control port for 125K counter (1F0)  
 CNTBAUD - Baud rate port for the timer associated with  
 the USART used for I/O (0F00)

The values in parentheses indicate the ports required by the Intel system 210, 220, and 230 models. Should the protocol be changed to support serial RS-232-C communications, as specified by the requirements for UNIB II, the commands associated with the USART will require modification. Currently, full duplex, one direction, asynchronous communications occurs between the UNIB and the host (25). Proper RS-232-C protocol calls for full duplex, synchronous communications. This will require implementing the following controls:

102	(AB) Signal Ground	(1)
105	(BA) Transmit Data	(2)
107	(BB) Receive Data	(3)
106	(CA) Request to Send	(4)
108	(CB) Clear to Send	(5)
107	(CC) Data Set Ready	(6)
112	(CD) Data Terminal Ready	(26)
109	(CF) Data Channel Receiver	(7)
114	(DD) Transmitter Signal Element Timing	(15)
115	(DC) Receiver Signal Element Timing	(17)

The brackets (1) indicate the RS-232-C pin corresponding to the signal. The parentheses indicate the VTA circuit destination. The number without brackets indicates the V.21, X.21, X.25 CCITT circuit destination. Currently all communications between the transport layer and the Internet layer occur using only three signals, which is the most straightforward hardware interface possible using a "RS-232-C protocol".

For CPU based machines, execution begins at vector address location 0000. For systems operating under ISIS, the start location must be calculated so that it does not interfere with system buffer



space. The start address may be as low as 0100H. While the code may begin at the above mentioned locations, the point of execution is usually at a different location (start with declarations associated with the code segment). After the program modules are compiled, linked, and loaded, the ISIS assembly function (under the program name MP2) gives the start location of the executable code. The ISIS operating system records this value automatically in the code segment and no further action needs to be taken. CP/M systems however require a jump instruction beginning at location 0100H to the actual point of execution. Hence the code segment must be located at location 0100H, a jump instruction added to locations 0100H - 0102H, and resaved using the "SAVE" command to form an executable "COM" file.

Transporting the software from ISIS to CP/M, as done with INSTLASH, requires the following procedure:

The file must be in the Intel HEX format. This comes from the ISIS function CPJHEX (part of) which converts an ISIS object file to an Intel HEX file. Available on single sided lists with the Intel 8080 system are software modules that first change Intel HEX format to an ISE HEX format, then change the ISE HEX format to CP/M format. The Intel HEX to ISE HEX format is invoked by file "hex" with the command form "hex filename T". The file name is filename without the drive path name as that is requested immediately after the program is invoked. The "T" indicates the menu selection for the Intel HEX to ISE HEX conversion. Other menu selections are available simply by typing "hex". Under this format of course a minimum boot of the CP/M system is required to edit the program. Conversion of the ISE HEX file to a CP/M file requires the

## Appendix B

### HOST SYSTEM MODIFICATION PROCEDURES

#### INTRODUCTION

Modifying the UMIB II host software requires three changes to be made. First, external calls to the existing operation system must be changed to call external procedures associated with the new operating system. Second, the I/O driver routines for the new serial I/O device must replace the previous I/O driver routines. Third, the software needs to be relocated to a point in system memory where execution is expected to take place. In replacing the BDOS calls in the assembly companion program (SBS.ASM in Childress' Thesis or HOST1.ASM in this Thesis) the public references to const, conin, conout, bdos, and exit may all be removed from the program file. At this point the entire file of external calls could be easily written in PL/I rather than assembly.

#### The Details

I/O driver routines implemented in Childress' SBS.ASM program support the MD 6250 universal asynchronous receiver/transmitter (UART). The I/O driver routines implemented in HOST1.ASM support the 1251/1251A USART. Due to the commonality between the two devices, the only change required in the software was proper initialization of the USART. Since many computer systems use the 1251/1251A as the serial interface device, the only change necessary to adapt the IS0 level routines to a new host would be in the I/O port addresses which are as follows:

SDATA - Serial data port (0F40)  
 SCONT - Serial control port (0F50)  
 SSTAT - Serial status port (0F60)  
 COUNTER - Mode control port for 8255 counter (0F30)  
 CNTBAUD - Baud rate port for the timer associated with  
 the USART used for I/O (0F00)

The values in parentheses indicate the ports required by the Intel system 110, 220, and 230 models. Should the protocol be changed to support serial RS-232-C communications, as specified by the requirements for UNID II, the commands associate with the USART will require modification. Currently, full duplex, one direction, asynchronous communications occurs between the UNID and the host (28). Proper RS-232-C protocol calls for full duplex, synchronous communications. This will require implementing the following controls:

102	(AD) Signal Ground	[1]
103	(BA) Transmit Data	[2]
104	(BB) Receive Data	[3]
105	(CA) Request to send	[4]
106	(CB) Clear to Send	[5]
107	(CC) Data Set Ready	[6]
108/2	(CD) Data Terminal Ready	[22]
109	(CF) Data Channel Received	[8]
114	(DE) Transmitter Signal Element Timing	[15]
115	(LC) Receiver Signal Element Timing	[17]

The brackets ([ ]) indicate the RS-232-C pin correspondance to the signal. The parentheses indicate the EIA circuit designation. The number without brackets indicates the V.24, X.21, X.25 CCITT circuit designation. Currently all communications between the transport layer and the internet layer occur using only three signals, which implement the simplest hardware interface possible using a "RS-232-C protocol".

For CP/M based machines, execution begins at memory address location 0100h. For systems operating under ISIS, the start location must be calculated so that it does not interfere with system buffer

space. The start address may be no lower in memory than 0100h. While the code may begin at the above mentioned locations, the point of execution is usually at a different location past the data declarations associated with the code segment. After the program modules are compiled, linked, and loaded, the ISIS memory map function (under file program.MP2) gives the start location of the executable code. The ISIS operating system records this value automatically in the code segment and no further action need be taken. CP/M systems however require a jump instruction beginning at location 0100h to the actual point of execution. Hence the code segment must be located at location 0103h, a jump instruction added to locations 0100h - 0102h, and resaved using the "SAVE" command to form an executable "COM" file.

Transporting the software from ISIS to CP/M, as done with HOSTLASH, requires the following procedure:

The file must be in the Intel HEX format. This comes from the ISIS function OBJHEX (pathname) which converts an ISIS object file to an Intel HEX file. Available on single sided disks with the Intel 286 system are software modules that first change Intel HEX format to an ISE HEX format, then change the ISE HEX format to CP/M format. The Intel HEX to ISE HEX format is invoked by file "icx" with the command form "icx filename f". The file name is given without the drive path name as that is requested immediately after the program is invoked. The "f" indicates the menu selection for the Intel HEX to ISE HEX conversion. Other menu selections are available simply by typing "icx". Under this form of program entry a warm boot of the CP/M system is required to exit the program. Conversion of the ISE HEX file to a CP/M file requires the

function "isc2" in the form "isc2 oldfilename newfilename". With this command the file created by the "icx" function is changed to the CP/M format and renamed to the second file name given. Both names may be the same. At this point a file exists in the CP/M format. That file will require the "load" ("MLOAD" under ZCPR) command to locate the file at the proper memory location of 0103h ("MLOAD" requires a bias of 3). Finally the file will require the assembly jump instruction to the point of execution (location 053Ch for HOST1.ASM). The instruction may be entered either in assembly code (j0 low byte high byte) or machine code (c3 low byte high byte).

#### Conclusions

From the above description, similar procedures may be developed for other operating systems, such as MSDOS, by adding a step that changes the CP/M formatted program to the target operating system format. The system is required to exit the program. Conversion of the ISE HEX file to a CP/M file requires the function "isc2" in the form "isc2 oldfilename newfilename". With this command the file created by the "icx" function is changed to the CP/M format and renamed to the second file name given. Both names may be the same. At this point a file exists in the CP/M format. That file will require the "load" ("MLOAD" under ZCPR) command to locate the file at the proper memory location of 0103h ("MLOAD" requires a bias of 3). Finally the file will require the assembly jump instruction to the point of execution (location 053Ch for HOST1.ASM). The instruction may be entered either in assembly code (j0 low byte high byte) or machine code (c3 low byte high byte).

function "isc2" in the form "isc2 oldfilename newfilename". With this command the file created by the "icx" function is changed to the CP/M format and renamed to the second file name given. Both names may be the same. At this point a file exists in the CP/M format. That file will require the "load" ("MLOAD" under ZCPR) command to locate the file at the proper memory location of 0103h ("MLOAD" requires a bias of 3). Finally the file will require the assembly jump instruction to the point of execution (location 055Ch for HOST1.ASM). The instruction may be entered either in assembly code (jmp low byte high byte) or machine code (c3 low byte high byte).

#### Conclusions

From the above description, similar procedures may be developed for other operating systems, such as MSDOS, by adding a step that changes the CP/M formatted program to the target operating system. For all systems is required to exit the program. Conversion of the ISE HEX file to a CP/M file requires the function "isc2" in the form "isc2 oldfilename newfilename". With this command the file created by the "icx" function is changed to the CP/M format and renamed to the second file name given. Both names may be the same. At this point a file exists in the CP/M format. That file will require the "load" ("MLOAD" under ZCPR) command to locate the file at the proper memory location of 0103h ("MLOAD" requires a bias of 3). Finally the file will require the assembly jump instruction to the point of execution (location 055Ch for HOST1.ASM). The instruction may be entered either in assembly code (jmp low byte high byte) or machine code (c3 low byte high byte).

## Appendix E

### UNIX Semaphores and Protected Regions

This appendix explains in detail the use of semaphores as implemented for the exchange of information between the SBC 544 and SBC 88/45 boards. This appendix originally comes from the previous work (15:Appendix E).

The use of semaphores is required to protect a critical region (20:77) of program execution from being disturbed by other concurrent processes in a multiprocess or multiprocessor environment. As was earlier noted, the UNIX I and II are multiprocessor and multiprocess systems. Data in the form of packets are exchanged between the two processors. The process on the SBC 544 board which alerts the SBC 88/45 board that a packet is ready for the SBC 88/45 process must ensure that the SBC 88/45 process is not manipulating the last set of data left by the SBC 544 process, otherwise the SBC 544 process will, in all likelihood, write over the old data with the new data and cause inadvertent destruction of desired data (20:77).

Equally as important, the SBC 88/45 process must ensure that the SBC 544 process is not manipulating data in the critical region of memory when the SBC 88/45 process wants access to the critical region. The method to ensure a writeover (race) condition does not exist often involves the use of P and V type semaphore operators. (20:83). The P and V operators are often implemented with low level hardware operations, such as TestAndSet which is called LOCK for the Intel processors, that will interrogate and set a flag to ensure that only one process is manipulating data in a critical region of memory at one time. Unfortun-

nately, as mentioned in Chapter Three of this thesis, the SEC 544 board does not have the TestAndSet (LOCK) capability even though the SBC 33/45 board does. The SEC 544 board also does not have the mechanisms to allow the SBC 33/45 board to use its LOCK operator. Therefore another method was devised which allows both boards to share certain portions of the common system memory while allowing only one processor and process to access this shared, critical region at one time, thus preventing inadvertent destruction of data (20:77). This method uses a variable with only two states, Ready and Done, as the semaphore. The SEC 544 process will check the variable for the Done state, and if Done, will update the packet pointer and set the semaphore to the Ready state. The SBC 33/45 process checks for the Ready state, and if Ready, will move the packet to another buffer for further processing and set the semaphore to the Done state. The SEC 544 process is allowed only to check the semaphore for Done and set the semaphore to Ready. The SBC 33/45 process is allowed only to check the semaphore for Ready and set the semaphore to Done. By implementing the processes in this manner, the processes will stay in synchronization and not manipulate data until the data is ready to be manipulated. Each process is not allowed to wait until the other process is completed; it will continue performing other tasks and will, at some later time, reenter the semaphore testing routine. This mechanism is illustrated in Figure E-1 with the aid of pseudocode.



```

IF DatagramAvailable then                (Process executed by SBC 544)
  If LSem = Done then
    Do;
    LPointer = .LocalReceive(NexttoSend);
    LSem = Ready;
    service(.LocalReceive(NexttoSend));
    end;

  If LSem = Ready then                    (Process executed by SBC 33/45)
    Do;
    move(LPointer, .NetworkTransmit(NextEmpty), PacketSize);
    LSem = Done;
    load(.NetworkTransmit(NextEmpty));
    end;

```

Figure E-1. Pseudocode for SBC 544 to SBC 33/45 Packet Movement

The first section of pseudocode corresponds to the process on the SBC 544 board and the second section of pseudocode corresponds to the concurrent process on the SBC 33/45 board. `DiagramAvailable` is an indication that a packet is available to move to the network board, `LPointer` is a pointer to the available packet, `LSem` is the semaphore, `NexttoSend` is the pointer to the next available entry for a packet in the SBC 544 memory, `NextEmpty` is the pointer to the next available entry for a packet in the SBC 33/45 memory. `LPointer` and `NextEmpty`, as well as the packet, are stored in the shared system memory. The routines `service` and `load` adjust the pointers within the tables given as arguments and `move` moves a specified amount of data from one location to a second location. The reader should recall that the tables `LocalReceive`, `LocalTransmit`, `NetworkReceive` and `NetworkTransmit` used in the FIVE software are circular FIFO queues whose first-in pointer is `NexttoSend` and first-out pointer is `NexttoSend`.

The above high-level code is divisible into smaller sets of high-

visible assembly language code, each of which can be interrupted by other processes on the respective SBC 544 or SBC 33/45 board. The sections of code that can interrupt the above processes, however, do not manipulate any data used by the above sections of code, and will therefore not disturb the critical sections except for inducing a non-critical time delay. In addition, the remaining processes on the board never manipulate the semaphore. The LocalReceive buffer is used by other SBC 544 processes. These are the host receive interrupt routine, where the table pointer NexttoEmpty is manipulated, and the send a datagram from local host to local host routine where the table pointer NexttoSend is manipulated. The latter routine, while manipulating the NexttoSend pointer, will only do so when the particular datagram is for local to local host movement. The particular routine of which the above code is a part interrogates the IP header of the first datagram pointed to by NexttoSend. If this datagram is for local host to local host movement, the datagram is moved and the NexttoSend pointer updated. However, if the datagram is for local host to network movement, the first section of pseudocode above is called. The pointer NexttoSend will only be updated if the SBC 33/45 is ready for another packet, otherwise the pointer is left untouched and the datagram in question is still at the top of the LocalReceive table waiting to be moved. So while the NexttoSend pointer can be manipulated by another process, it is done so only if the datagram in question at the top of the LocalReceive table is going back to a local host and not to the network. Therefore the NexttoSend pointer will be updated for a network destined packet only if the datagram in question at the top of the LocalReceive

table is going to the network when the first section of the pseudocode is entered.

It should be noted that while both processors have the ability to access the shared system memory at the same time, and will probably attempt to do so, they cannot, in fact, read (or write) to the same shared location at exactly the same instant. This 'read at the same instant' phenomena is prevented by the hardware design of both boards. If the SBC 544 processor is in the middle of a fetch from shared memory, the hardware design locks out access to the memory to other processors with access to that shared memory. Therefore, the SBC 33/45 board cannot access the desired location until the SBC 544 board has completed its current instruction, whereupon, the SBC 33/45 may then access the shared memory. The same holds true for an access of shared memory by the SBC 33/45 board. When the SBC 33/45 processor accesses shared memory, other processors are locked out from accessing the same shared memory until the SBC 33/45 has completed its current instruction. Therefore, if the SBC 33/45 is executing the instructions to set 'LSeM = Done', and the SBC 544 is fetching the 'LSeM' location to interrogate for 'Done', there will not be a simultaneous access of the location 'LSeM' as explained above. Therefore the SBC 544 interrogation of 'LSeM' will find its value either 'Done' or 'Ready' as expected and execute the critical section accordingly.

The process communication from the SBC 33/45 board to the SBC 544 board is identical to that for the SBC 544 to SBC 33/45 board explained above. The variable names are different so as to keep the two processes and functions separated. The two processes function the same and the

discussion above applies to the SBC 30/45 to SBC 544 board communication. The pseudocode for the SBC 30/45 to SBC 544 board communication is shown in Figure E-2.

```

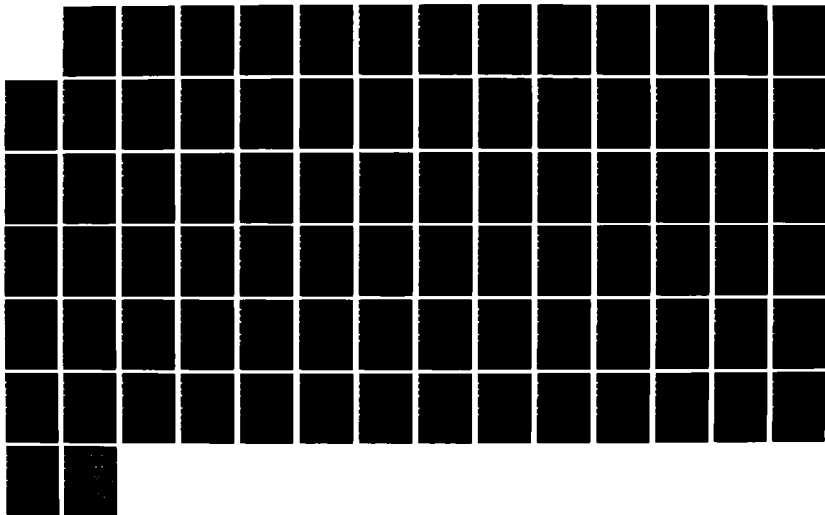
If PacketAvailable then                (Process executed by SBC 30/45)
  If NSev = Done then
    Go;
    NPointer = .NetworkReceive(NextToSend)
    NSev = Ready
    service(.NetworkReceive(NextToSend))
  end;

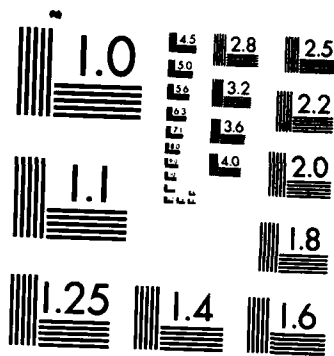
If NSev = Ready then                   (Process executed by SBC 544)
  Go;
  Move(NPointer, .NetworkReceive(NextToSend), DataTransferSize)
  NSev = Done
  Load(.LocalTransmit(NextEmpty))
end;

```

Figure E-2. Pseudocode for SBC 30/45 to SBC 544 Packet Movement

AD-A164 076      DEVELOPMENT AND IMPLEMENTATION OF THE X25 PROTOCOL FOR      3/3  
THE UNIVERSAL NETW. (U) AIR FORCE INST OF TECH  
WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI.      M W WEBER  
UNCLASSIFIED      DEC 85 AFIT/GE/ENG/85D-52-VOL-1      F/G 9/2      ML





MICROCOPY RESOLUTION TEST CHART  
NATIONAL BUREAU OF STANDARDS-1963-A

## Appendix F

### Transmit Request/Transmit Acknowledge Handshake

This appendix explains the details of the transmit request/transmit acknowledge mechanism implemented in the SSC 544 local host software. The appendix comes from the previous work (15:Appendix F).

The transmit request/transmit acknowledge mechanism is used to synchronize the UNIB II network layer software with a comparable process on a local host. The mechanism allows the orderly transmission of datagrams between the host and the UNIB II. This allows the UNIB II to reliably send and receive datagrams from a host that is slower than the speed of the UNIB II, whether the host polls the receive port or has a slow interrupt response, or a host that does not have an interrupt driven receive port from the UNIB II. This mechanism, or something similar such as the DTR-DSR or RTS-CTS hardware handshake, must be implemented to allow the orderly transmission of datagrams between the UNIB II and its connected hosts. The particular implementation explained below was chosen to accommodate the NETOS (LSI-11) network in the DELNET, which uses a software transmit request/transmit acknowledge scheme. The NETOS does not use a hardware handshake such as the RTS-CTS because the signals are not implemented on that system. Other mechanisms, such as KON-XOFF, may be relatively easily implemented in the UNIB II software to accommodate similar the mechanisms in other networks.

The particular mechanism used by the NETOS can be described as follows (21). When a node in the NETOS desires to send a packet to another node, it first sends a transmit request (TR) to the desired

node. The receiving node then, when it recognizes a TR was sent to it, will send a transmit acknowledge (TA) back to the sender when it is ready to receive a packet. The sending node, when it recognizes the TA from the receiving node, sends the datagram to the receiving node. There is no final acknowledge sent by the receiver back to the sender to acknowledge the reception of the datagram. The TR/TA mechanism effectively reduces a normally full duplex channel to a half duplex channel. The process can be diagrammed as shown in Figure F-1.

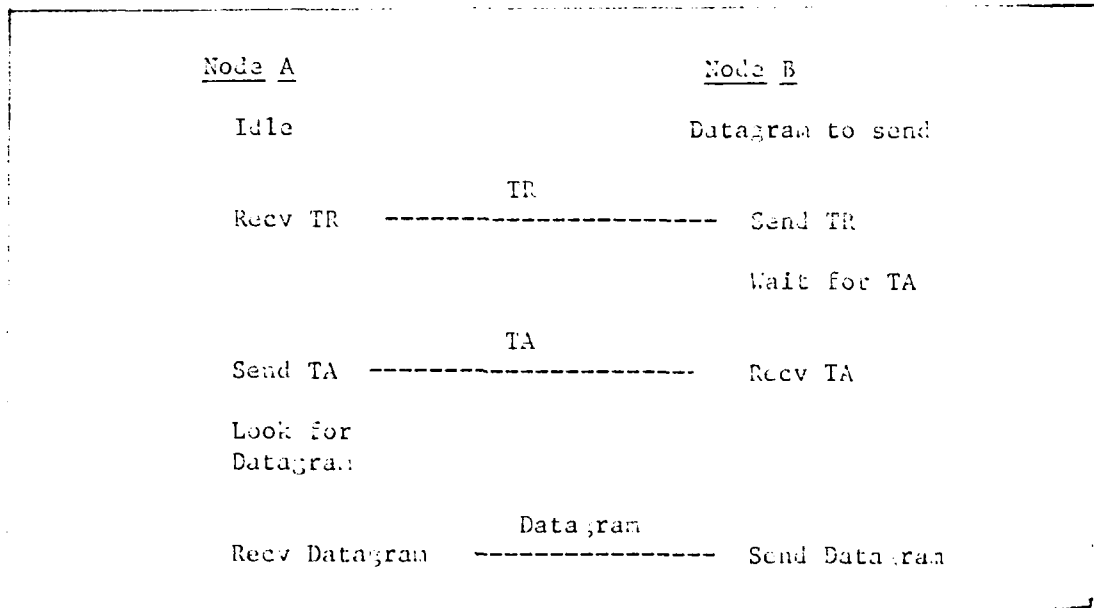


Figure F-1. NETOS Transmit Request/Transmit Acknowledge Handshake.

The mechanism is implemented in the UNID II software with four boolean flags for each host port. Four flags are required since both the NETOS node and the UNID will send and receive datagrams using the TR/TA mechanism, providing a full handshake for each direction. The UNID must know which state it is in so that it can communicate correctly with the NETOS node. The four flags are Transmit transmit request



(TXTR), Receive transmit acknowledge (RXTA), Receive transmit request (RXTR), and Transmit transmit acknowledge (TXTA). Each flag has the value TRUE or FALSE. The initial state is all four flags FALSE. Of the 16 possible states, the five allowed states are shown in Figure F-2. A '0' represents FALSE and a '1' represents TRUE.

<u>TXTA</u>	<u>RXTA</u>	<u>RXTR</u>	<u>TXTR</u>	
0	0	0	0	Initial state
1	0	0	0	Datagram to send
1	1	0	0	OK to send datagram
1	1	0	0	Send the datagram
0	0	0	0	Reset the flags after sending datagram
0	0	1	0	Datagram receive request
0	0	1	1	OK to receive datagram
0	0	1	1	Receive datagram
0	0	0	0	Reset flags after receiving datagram

Figure F-2. UNID TR/TA Allowable States.

Note that the mechanism starts in the all zero, or FALSE, state with no datagrams to send or receive and returns to the all zero state at the completion of sending or receiving a datagram. Also, only one process of sending a datagram or receiving a datagram is allowed at one time. While this is a requirement for the NETOS, and possibly other networks, the UNID software is totally interrupt driven and can send and receive a datagram simultaneously without this handshake mechanism. The UNID II local software on the 544 board has been designed through the

use of a boolean flag, labeled TRTA so that the TR/TA half duplex handshake may be used or not used on any given host port of the 544 board. The entire handshake process may also be represented in a state diagram, as shown in Figure F-3.

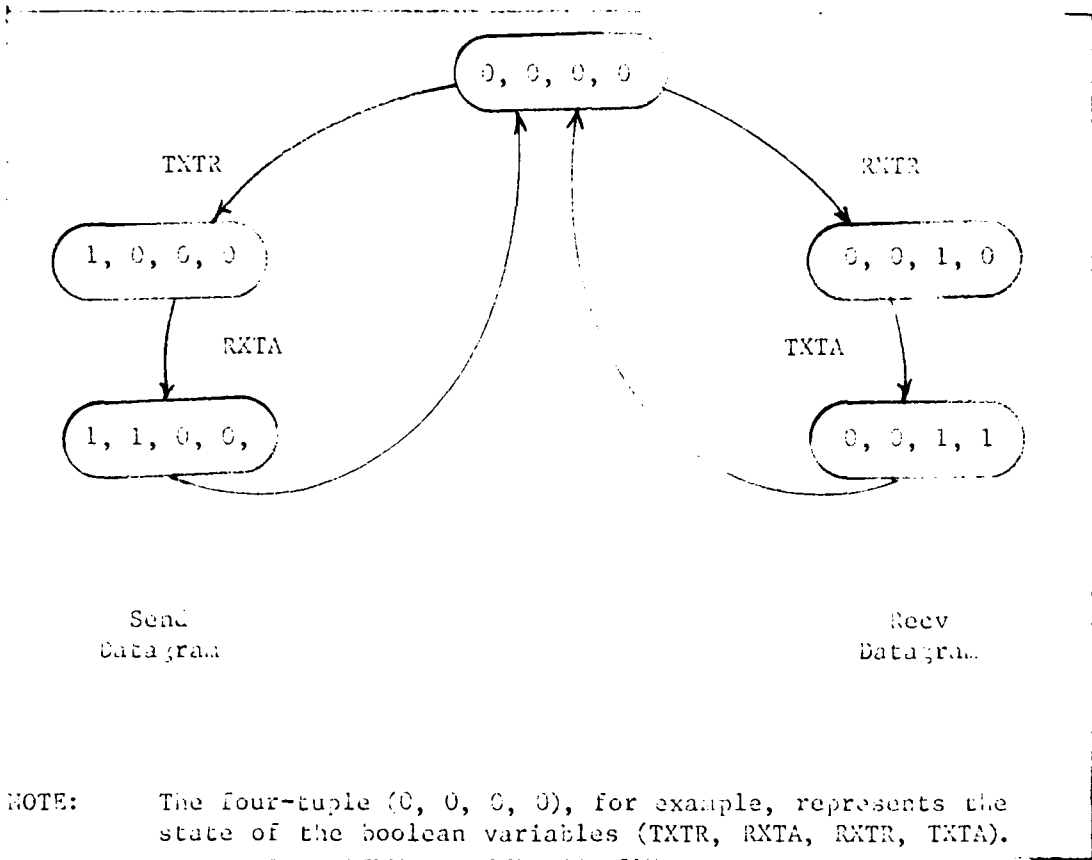


Figure F-3. State Diagram of the TXTR Handshake.

## Appendix C

### DELNET/UNIX Header Information

This appendix expands upon the TCP/IP datagram, packet, and frame header information presented briefly in Chapter One (93:Appendix C). Each byte in a complete datagram, packet, and frame is shown with the appropriate bit information within each byte. The name of each byte position, along with the array index number, is given for each byte. The subscripts H and L refer to most significant byte and least significant byte, respectively. Similarly, the subscripts 3, 2, and 1 indicate the most significant byte, the next significant byte and the least significant byte, respectively. The contents of each byte conform to the standards established in (94, 95, 93:Appendix C, 115). Entries that contain letters refer to specific bits that must be initialized or set according to how and when they are used. For example, the packet source address will be a constant that depends upon the particular UNIX and port number to which a host is connected. The Type of Service byte on page D-2 depends upon the precedence, delay, throughput and routing required by the transport and higher level protocols. Those bits and bytes that contain letters are variables that are data and user dependent. Those bytes that are empty are not yet used by the UNIX II software and are currently filled with zeros. The particular mapping shown was used for the testing of the UNIX II in the LSI-11 NETOS test. Each byte is further explained in (94, 95, 93:Appendix C, 115).

Name	Index		Bits			
	Frame	Packet	MSB			LSB
			0	0	0	0
Address: LAP B * x = 1 or 0	0		0	0	0	0
Control b = poll/final bit c = control format for frame	1		c	c	c	c
GFI/LGCN g = 0001 h for datagram service l = defaults to 0 h	2	0	0	0	0	0
LCN u = UMID #, c = channel #	3	1	u	u	u	u
Sequence #	4	2	n	n	n	n
Source/Destination Length s = source d = destination	5	3	s	s	s	s
Source Address u = umid # c = channel #	6	4	u	u	u	u
Destination Addr u = umid # c = channel #	7	5	u	u	u	u
Padding	8	6	0	0	0	0
Facility Length l = length bit	9	7	0	0	1	1
Facility Code (set to 0)	10	8	0	0	0	0
Facility Parameter (set to 0)	11	9	0	0	0	0

Name	Frame	Index		MSB	Bits		LSB
		Packet	Data,gram				
IP Header	12-43	10-41	0-31				
Vers #/HLength	12	10	0	0 1 0 0	1 0 0 0		
Type of Service	13	11	1	p p p p	t r 0 0		
<p>p = precedence  p = 000 = routine Data,gram  p = 110 = Internet Control</p>							
Total Length(H)	14	12	2	0 0 0 0	0 0 0 0		
Total Length(L)	15	13	3	1 0 0 0	0 0 0 0		
User Ident(H)	16	14	4	u u u u	u u u u		
User Ident(L)	17	15	5	u u u u	u u u u		
Flags/Frag. Off(H)	18	16	6	0 1 0	0 0 0 0 0		
Fragment Offset(L)	19	17	7	0 0 0 0	0 0 0 0		
Time to Live (60)	20	18	8	0 0 1 1	1 1 0 0		
Protocol	21	19	9	0 0 0 0	0 1 1 0		
Header Checksum(H)	22	20	10	0 0 0 0	0 0 0 0		
Header Checksum(L)	23	21	11	0 0 0 0	0 0 0 0		
Source Address:							
Control/Country	24	22	12	0 0 0 0	1 0 0 1		
Network Code(UNID #) Host Code(H)	25	23	13	0 0 1 0	h h h h		
Host Code(L)/ Port Code(2)	26	24	14	h h h h	0 p p p		
Port Code(1)/ Port Code(0)	27	25	15	p p p p	p p p p		
Destination Address:							
Control/Country	28	26	16	0 0 0 0	1 0 0 1		
Network Code(UNID #) Host Code(H)	29	27	17	n n n n	h h h h		

Name	Frame	Index		BITS			
		Packet	Data,gram	MSB			LSB
Host Code(L)/ Port Code(2)	30	20	18	0	0	0	0
Port Code(1)/ Port Code(0)	31	29	19	0	0	0	0
Security(H)	32	30	20	1	0	0	0
Security(L)	33	31	21	0	0	0	0
S Field(H)	34	32	22				
S Field(L)	35	33	23				
C Field(H)	36	34	24				
C Field(L)	37	35	25				
H Field(H)	38	36	26				
H Field(L)	39	37	27				
TCC Field(2)	40	38	28				
TCC Field(1)	41	39	29				
TCC Field(0)	42	40	30				
UF Padding	43	41	31	0	0	0	0
TCP Header	44-67	42-65	32-55				
Source Port(H)	44	42	32				
Source Port(L)	45	43	33				
Destin Port(H)	46	44	34				
Destin Port(L)	47	45	35				
Sequence #(3)	48	46	36				
Sequence #(2)	49	47	37				
Sequence #(1)	50	48	38				
Sequence #(0)	51	49	39				

Name	Frame	Index		Datagram	Bits										
		Packet			MSB							LSB			
Acknowledge #(3)	52	50		40											
Acknowledge #(2)	53	51		41											
Acknowledge #(1)	54	52		42											
Acknowledge #(0)	55	53		43											
Data Offset/Resv	56	54		44	0	1	1	0	0	0	0	0			
Reserved/Control	57	55		45	0	0	u	a	p	r	s	f			
Window(I)	58	56		46											
Window(L)	59	57		47											
Checksum(H)	60	58		48											
Checksum(L)	61	59		49											
Urgent Ptr(H)	62	60		50											
Urgent Ptr(L)	63	61		51											
Option	64	62		52											
Padding(2)	65	63		53	0	0	0	0	0	0	0	0			
Padding(1)	66	64		54	0	0	0	0	0	0	0	0			
Padding(0)	67	65		55	0	0	0	0	0	0	0	0			
User Data	68-139	66-137		56-127	x	x	x	x	x	x	x	x			

Figure G-1. DELNET/UNID Detailed Header Information

Appendix H

Data Dictionary

This appendix contains the data dictionary updated from (15:Appendix H) for the four programs that comprise the network and data link layer simulations, the validation and test programs used with the UNID II and NETOS, and the ISIS host simulation software. The simulation dictionaries are presented first followed by the dictionary for the software on the SBC 544 and the CP/M system. Each of the four programs has its own subdictionary which contains a section for constants, variables, and procedures. Each entry is listed in alphabetical order.

The batch files used to link and locate the object code generated by the compiler are also included at the end of each applicable subdictionary.

The appendix is subdivided into four subdictionaries which are listed as follows:

<u>Subdictionary</u>	<u>Page</u>
1. Network Layer Simulation . . . . .	H-2
2. Data Link Layer Simulation . . . . .	H-7
3. SBC 544 Validation . . . . .	H-15
4. ISIS Host. . . . .	H-19



## 1. Network Layer Simulation

The purpose of this program is to simulate the network layer software on the Intel Software Development System.

### Constants

ASCII(\*) - Array of ASCII characters used for converting binary to hex and hex to binary numbers for display on the console.

DATA\$GRAM\$SIZE - Number of bytes in a datagram (128) received from a host.

DATA\$TABLE\$SIZE - Number of bytes within a data table.

L\$RI\$DEST\$ERR - Local route in destination error.

L\$RO\$DEST\$ERR - Local route out destination error.

MAX\$COUNTRY\$CODE - Maximum number of countries operational on the DELNET.

MAX\$NETWORK\$CODE - Maximum number of UNIDs operational within a particular country.

PACKET\$SIZE - Number of bytes in a packet (138).

PACKET\$SIN\$TABLE - Number of packets in a packet table.

PACKET\$TABLE\$SIZE - Number of bytes in a packet table.

R\$CONN - I/O handle number for ISIS console call.

STAT\$NBR - Number of the status entries to be included in the status table.

SYSSMEM\$BASE - Base address used to locate the shared table and variables.

SYSS\$BASE - Base label used to properly locate the shared table and variables. Used with SYSSMEM\$BASE.

THIS\$COUNTRY\$CODE - Unique code indicating in which country THIS\$UNID\$NBR resides.

THIS\$UNID\$NBR - Unique UNID number for the UNID performing the interface between local hosts and the DELNET.

TCP\$DATA\$SIZE - Number of user data bytes in the TCP header.

TA - Transmit acknowledge character.  
TR - Transmit request character.  
W\$CONN - I/O handle number for ISIS console call.

Variables

ACTUAL - Number of characters returned from ISIS console read call.  
BUFFER - 128 byte buffer used with ISIS console read call.  
DESTINATION - Indicates whether a received datagram is destined for the network or another attached local host.  
DESTINATION\$ADDRESS - Indicates the destination address of a datagram.  
SOURCE\$ADDRESS - Indicates the source address of a datagram.  
ERRNUM - Number of the error returned from an ISIS system call.  
LC01NE - Pointer in the array LC01TB pointing to the next available position for a received datagram.  
LC02NE - Pointer in the array LC02TB pointing to the next available position for a received datagram.  
LC03NE - Pointer in the array LC03TB pointing to the next available position for a received datagram.  
LC04NE - Pointer in the array LC04TB pointing to the next available position for a received datagram.  
LC01NS - Pointer in the array LC01TB pointing to the next datagram to service.  
LC02NS - Pointer in the array LC02TB pointing to the next datagram to service.  
LC03NS - Pointer in the array LC03TB pointing to the next datagram to service.  
LC04NS - Pointer in the array LC04TB pointing to the next datagram to service.  
LC01SZ - The maximum number of bytes in the LC01TB array.  
LC02SZ - The maximum number of bytes in the LC02TB array.  
LC03SZ - The maximum number of bytes in the LC03TB array.

LC04SZ - The maximum number of bytes in the LC04TB array.

LC01TB - Local receive table for host port number one.

LC02TB - Local receive table for host port number two.

LC03TB - Local receive table for host port number three.

LC04TB - Local receive table for host port number four.

LPTR\$1, LPTR\$2, LPTR\$3, LPTR\$4 - Pointer to the current packet to be passed to the data link layer.

LSEM\$1, LSEM\$2, LSEM\$3, LSEM\$4 - Semaphore used by the network and data link layers to indicate the state of the packet transfer.

LSPARE\$1, LSPARE\$2, LSPARE\$3, LSPARE\$4 - Spare memory locations used by the SBC 88/45 for it's pointer transfer.

NPTR\$1, NPTR\$2, NPTR\$3, LPTR\$4 - Pointer to the current packet to be passed to the network layer.

NSEM\$1, NSEM\$2, NSEM\$3, NSEM\$4 - Semaphore used by the network and data link layers to indicate the state of the packet transfer.

NSPARE\$1, NSPARE\$2, NSPARE\$3, NSPARE\$4 - Spare memory locations used by the SBC 88/45 for it's pointer transfer.

TX01NE - Pointer in the array TX01TB pointing to the next available position for a transmitted datagram.

TX02NE - Pointer in the array TX02TB pointing to the next available position for a transmitted datagram.

TX03NE - Pointer in the array TX03TB pointing to the next available position for a transmitted datagram.

TX04NE - Pointer in the array TX04TB pointing to the next available position for a transmitted datagram.

TX01NS - Pointer in the array TX01TB pointing to the next datagram to service.

TX02NS - Pointer in the array TX02TB pointing to the next datagram to service.

TX03NS - Pointer in the array TX03TB pointing to the next datagram to service.

TX04NS - Pointer in the array TX04TB pointing to the next datagram to service.

TX01SZ - The maximum number of bytes in the TX01TB array.

TX02SZ - The maximum number of bytes in the TX02TB array.

TX03SZ - The maximum number of bytes in the TX03TB array.

TX04SZ - The maximum number of bytes in the TX04TB array.

TX01TB - Local receive table for host port number one.

TX02TB - Local receive table for host port number one.

TX03TB - Local receive table for host port number one.

TX04TB - Local receive table for host port number one.

MESSAGE(\*) - Test message array.

STATUS - Error status of ISIS console I/O calls.

#### Procedures

DET\$ADDR - Determine the destination of the datagram from the attached host.

DET\$ADDR\$NL - Determine the destination of the datagram passed from the data link layer.

ERROR - I/O error handler for ISIS operating system calls.

EXIT - Graceful method to end the simulation; returns to the ISIS operating system.

INIT - Initializes the variables to their initial states.

INIT\$TAB - Initializes the network and data link layer tables and pointers to their initial values.

LD\$TAB\$H\$SKP - Housekeep a specified buffer table load pointer.

LOOP - Simulates the semaphore check and set operation of the SBC 88/45 board to turn a frame around to the network layer.

MOVETO\$LOCAL - Move a datagram from a receive host buffer or the data link layer buffer to the local host transmit buffer.

READ - Read a line of character input from the console; an ISIS operating system call.

ROUTE\$IN - Route received datagrams from the local hosts to the data link layer or the local host transmit buffers.

ROUTE\$OUT - Send the datagrams in the transmit buffers to the local hosts.

SEND\$PACKET - Transforming the user datagram into a packet for transfer to the data link layer.

SERVICE\$LOOP - Turns a frame around at the data link layer. The source and destination headers are exchanged.

SET\$TRTA - Queries operator for which host channels will use the TRTA handshake.

SNDBSEQ - Takes a message string from the calling procedure and outputs it to the ISIS operating system.

SRVC\$TAB\$HSKP - Housekeep a specified buffer table service pointer.

WRITE - Write a line of character information to the console; an ISIS operating system call.

Link and Locate Batch File (LNK544.CSD)

CAUTION: Do not change address or other parameters in the following batch file. They are highly hardware dependent on the System III and the ISIS operating system.

```
LINK SIM544.OBJ,SYSTEM.LIB,PLM80.LIB TO SIM544.LNK MAP
LOCATE SIM544.LNK TO SIM544 STACKSIZE(100H) ORDER(CODE,DATA,&
STACK,MEMORY) CODE(5000H) MAP PRINT(SIM544.MP2)
TYPE SIM544.MP2
```

## 2. Data Link Layer Simulation

The purpose of this program is to simulate the data link layer software on the Intel Software Development System.

### Constants

A\$ADD - LAP B address byte for command and response frames.

ASCII(\*) - Array of ASCII characters used for converting binary to hex and hex to binary numbers for display on the console.

B\$ADD - LAP B address byte for command and response frames.

CMDR\$FRAME\$SIZE - Size of Command Reject Frames in bytes (3).

CONCTC - Network monitor counter timer port address.

CONCMD - Network monitor USART command port address.

CONDAT - Network monitor USART data port address.

DATA\$GRAM\$SIZE - Number of bytes in a datagram (128) received from a host.

DATA\$TABLE\$SIZE - Number of bytes within a data table.

DISC\$CNTL - Disconnect frame mask byte.

I\$CNTL - Information frame mask byte.

I\$FRAME\$SIZE - Size of Information frames in bytes (140).

L\$RI\$DEST\$ERR - Local route in destination error.

L\$RO\$DEST\$ERR - Local route out destination error.

MAX\$COUNTRY\$CODE - Maximum number of countries operational on the DELNET.

MAX\$NETWORK\$CODE - Maximum number of UNIDs operational within a particular country.

MAXNOA - Maximum number of timing counts for network channel A.

MAXNOB - Maximum number of timing counts for network channel B.

MAXRETRANS\$A - Maximum number of retransmissions of a frame for network channel A.

MAXRETRANS\$B - Maximum number of retransmissions of a frame for network channel B.

PACKET\$SIZE - Number of bytes in a packet (133).

PACKETS\$IN\$TABLE - Number of packets in a packet table.

PACKET\$TABLE\$SIZE - Number of bytes in a packet table.

P\$BIT\$MASK - Mask for poll/final bit.

R\$CONN - I/O handle number for ISIS console call.

S\$FRAME\$SIZE - Size of Supervisory frames in bytes (2).

SABM\$CNTL - Set asynchronous balanced mode maske byte.

STAT\$NBR - Number of the status entries to be included in the status table.

THIS\$COUNTRY\$CODE - Unique code indicating in which country THIS\$UNID\$NBR resides.

THIS\$UNID\$NBR - Unique UNID number for the UNID performing the interface between local hosts and the DELNET.

TCP\$DATA\$SIZE - Number of user data bytes in the TCP header.

U\$FRAME\$SIZE - Size of unnumbered frames in bytes (2).

UA\$CNTL - Unnumbered Acknowledgment mask byte.

W\$CONN - I/O handle number for ISIS console call.

#### Variables

ACTUAL - Number of characters returned from ISIS console read call.

BUFFER - 128 byte buffer used with ISIS console read call.

CTCNOA - Progressive number of time counts for network channel A.

CTCNOB - Progressive number of time counts for network channel B.

DESTINATION - Indicates whether a received datagram is destined for the network or another attached local host.

SOURCE\$ADDRESS - Indicates the source address of a datagram.

DM\$MODE - Indicates when system in the Disconnect Mode of operation.

ERRNUM - Number of the error returned from an ISIS system call.

I\$FRAME\$QUE - The I frame queue for I frames being sent down to the physical layer.

I\$FRAME\$QUE\$NE - The next available pointer for the I\$FRAME\$QUE.

I\$FRAME\$QUE\$NS - The next to send pointer for the I\$FRAME\$QUE.

I\$FRAME\$QUE\$SZ - The size of the array for I\$FRAME\$QUE.

LCNTNE - Pointer in the array LCNTTB pointing to the next available position for a received datagram.

LCNTNS - Pointer in the array LCNTTB pointing to the next datagram to service.

LCNTSZ - The maximum number of bytes in the LCNTTB array.

LCNTTB - Local to network table.

L\$PTR\$1 - Pointer to host channel 1 for datagram to send to the data link.

L\$PTR\$2 - Pointer to host channel 2 for datagram to send to the data link.

L\$PTR\$3 - Pointer to host channel 3 for datagram to send to the data link.

L\$PTR\$4 - Pointer to host channel 4 for datagram to send to the data link.

L\$SEM\$1 - Semaphore for host channel 1.

L\$SEM\$2 - Semaphore for host channel 2.

L\$SEM\$3 - Semaphore for host channel 3.

L\$SEM\$4 - Semaphore for host channel 4.

L\$SPARE\$1 - Unused semaphre for channel 1.

L\$SPARE\$2 - Unused semaphre for channel 2.

L\$SPARE\$3 - Unused semaphre for channel 3.

L\$SPARE\$4 - Unused semaphre for channel 4.



NTLCNE - Pointer in the array NTLCTB pointing to the next available position for a received datagram.

NT01NE - Pointer in the array NT01TB pointing to the next available position for a received datagram.

NT02NE - Pointer in the array NT02TB pointing to the next available position for a received datagram.

NTLCNS - Pointer in the array NTLCTB pointing to the next datagram to service.

NT01NS - Pointer in the array NT01TB pointing to the next datagram to service.

NT02NS - Pointer in the array NT02TB pointing to the next datagram to service.

NTLCSZ - The maximum number of bytes in the NTLCTB array.

NT01SZ - The maximum number of bytes in the NT01TB array.

NT02SZ - The maximum number of bytes in the NT02TB array.

NTLCTB - Local receive table for host port number two.

NT01TB - Local receive table for host port number three.

NT02TB - Local receive table for host port number four.

MESSAGE(\*) - Test message array.

N\$PTR\$1 - Pointer to current frame to send to packet layer from channel A.

N\$PTR\$2 - Pointer to current frame to send to packet layer from channel B.

N\$SEMS\$1 - Data link semaphore for channel A.

N\$SEMS\$2 - Data link semaphore for channel B.

N\$PARE\$1 - Currently unused spare semaphore for data link tables.

N\$PARE\$2 - Currently unused spare semaphore for data link tables.

TX01NE - Pointer in the array NT01TX pointing to the next available position for a transmitted datagram.

RCV\$STATE\$A - State variable N(R) for channel A.

RCV\$STATE\$B - State variable N(R) for channel B.

RETRANS\$A - Progressive number of retransmissions of a frame for channel A.

RETRANS\$B - Progressive number of retransmissions of a frame for channel A.

RNR\$MODE\$A - Indicates when channel A is in the receive not ready mode of operation.

RNR\$MODE\$B - Indicates when channel B is in the receive not ready mode of operation.

SABM\$MODE\$A - Indicates when channel A is in the SABM state.

SABM\$MODE\$B - Indicates when channel B is in the SABM state.

SEND\$STATE\$A - State variable N(S) for channel A.

SEND\$STATE\$B - State variable N(S) for channel B.

SEQ\$BIT\$A - Frame acknowledge bit for channel A.

SEQ\$BIT\$B - Frame acknowledge bit for channel B.

SEQNUM\$A - Sequence number for the received channel A sequence number.

SEQNUM\$B - Sequence number for the received channel B sequence number.

STATUS - Error status of ISIS console I/O calls.

SYSBASE - Current base for location of embedded operational code.

THIS\$SEQ\$BIT\$A - Current sequence bit for frame to transmit in channel A.

THIS\$SEQ\$BIT\$B - Current sequence bit for frame to transmit in channel B.

TIMCHA - Current time count for channel A.

TIMCHB - Current time count for channel B.

UA\$ACK\$CHA - Indicates when an unnumbered acknowledgement is received for channel A.

UA\$ACK\$CHB - Indicates when an unnumbered acknowledgement is received for channel B.

U\$CMD\$QUE\$A - Contains the oldest unacknowledged unnumbered command for channel A.

U\$CMD\$QUE\$B - Contains the oldest unacknowledged unnumbered command for channel B.

Procedures

MODULE NETX25:

DQ\$DECODE\$EXCEPTION - External ISIS call to decode error exceptions.

DQ\$CLOSE - External ISIS call to close an I/O handle.

DQ\$DETACH - External ISIS call to detach an I/O device.

DQ\$EXIT - External ISIS call to exit the current program back to the  
ISIS operating system.

DQ\$ATTACH - External ISIS call to attach an I/O device.

DQ\$CREATE - External ISIS call to obtain an I/O handle.

DQ\$OPEN - External ISIS call to open a file.

DQ\$READ - External ISIS call to read an opened file.

DQ\$WRITE - External ISIS call to write an opened file.

INIT - Initializes the variables to their initial states.

INIT\$TAB - Initializes the network and data link layer tables and  
pointers to their initial values.

LD\$TAB\$H\$SKP - Housekeep a specified buffer table load pointer.

LOOP - Simulates the operation of another UNID in the network.

ROUTE\$IN - Route received packets and frames from the network  
layer and the network.

ROUTE\$OUT - Send the frames to the network or packets to the network  
layer.

BUILD\$I\$PACKET - Transforms the user packet into a frame for transfer  
to the network.

SERVICE\$LOOP - Turns a frame around in the network. The source  
and destination headers are exchanged.

SNDSEQ - Takes a message string from the calling procedure and outputs  
it to the ISIS operating system.

SRVC\$TAB\$H\$SKP - Housekeep a specified buffer table service pointer.

MODULE LAPO:

DSPLY\$FRAME\$HDR - Displays the frame header for specified transmit or receive tables.

FIND\$U\$CMD - Finds the outstanding unnumbered command in the U\$CMD\$QUE table.

PRINTI - Prints an integer when given a byte value.

RCV\$CMDR Procedure to receive CMDR frames.

RCV\$DISC - Procedure to receive DISC frames.

RCV\$DM - Procedure to receive DM frames.

RCV\$I\$FRAME - Procedure to receive I Frames.

RCV\$REJ - Procedure to receive REJ frames.

RCV\$RNR - Procedure to receive RNR frames.

RCV\$RR - Procedure to receive RR frames.

RCV\$SABM - Procedure to receive SABM frames.

RCV\$UA - Procedure to receive UA frames.

SND\$CMDR Procedure to send CMDR frames.

SND\$DISC - Procedure to send DISC frames.

SND\$DM - Procedure to send DM frames.

SND\$I\$FRAME - Procedure to send I Frames.

SND\$REJ - Procedure to send REJ frames.

SND\$RNR - Procedure to send RNR frames.

SND\$RR - Procedure to send RR frames.

SND\$SABM - Procedure to send SABM frames.

SND\$UA - Procedure to send UA frames.

MODULE LAPB1:

DET\$ADDR - Determine the destination of the packet from the attached host.

DET\$DEST\$ONE - Determines the destination for frames in channel A.

DET\$DEST\$TWO - Determines the destination for frames in channel B.

DET\$DEST\$LN - Determines the local host destination for a frame going to a local host.

READTAB - Reads the network to local table and displays to the console output device.

TIMES\$DELAY\$CHA - Time out mechanism for channel A.

TIMES\$DELAY\$CHB - Time out mechanism for channel B.

MODULE PCKT:

ROUTE\$PACKET - Moves a frame to the appropriate local receive table.

Link and Locate Batch File (netx25.csd)

CAUTION: Do not change address or other parameters in the following batch file. They are highly hardware dependent on the System III architecture and the ISIS operating system.

```
run link86 netx25.obj, lapb0.obj, lapb1.obj, pkt.obj, small.lib
run loc86 netx25.lnk ad(sm(code(7800h),const(a500h),data(c400h), &
stack(ed00h),memory(f300h),??seg(f200h)))
```

### 3. SBC 544 Validation

The purpose of this program is to operate the network layer software on the Intel SBC 544. All the constants, variables and procedures from the network layer simulation are used in this program.

#### Constants

BRF0, BRFl, BRf2, BRf3 - Data rate factor, USART 0, 1, 2, and 3.

SIM\$MASK - Set interrupt mask mask.

MASTER - Port number for Master Mode.

SLAVE - Port number for Slave Mode.

#### 8251A USART Constants:

US\$PO\$CMD - SERIAL PORT 0 COMMAND  
US\$PO\$STAT - SERIAL PORT 0 STATUS  
US\$PO\$DATA - SERIAL PORT 0 DATA  
US\$P1\$CMD - SERIAL PORT 1 COMMAND  
US\$P1\$STAT - SERIAL PORT 1 STATUS  
US\$P1\$DATA - SERIAL PORT 1 DATA  
US\$P2\$CMD - SERIAL PORT 2 COMMAND  
US\$P2\$STAT - SERIAL PORT 2 STATUS  
US\$P2\$DATA - SERIAL PORT 2 DATA  
US\$P3\$CMD - SERIAL PORT 3 COMMAND  
US\$P3\$STAT - SERIAL PORT 3 STATUS  
US\$P3\$DATA - SERIAL PORT 3 DATA  
US\$MODE - SERIAL PORT MODE  
US\$COMMAND - SERIAL PORT COMMAND  
US\$RESET\$CMD - RESET USART  
US\$DTR\$ON - RTS, RXE, DTR, TXE  
US\$CRT\$CMD - RTS, ER, RXE, DTR, TXE  
US\$TTY\$CMD - RTS, ER, RXE, TXE  
US\$DTR\$OFF - RTS, RXE, TXE  
US\$RXRDY - RECIEVER READY  
US\$TXE - TRANSMITTER EMPTY  
US\$TXRDY - TRANSMITTER READY  
PARITY\$MASK - MASK OFF PARITY BIT

#### 8253 Interval Timer Constants:

ITI\$CONT - INTERVAL TIMER 1 CONTROL  
ITI\$CNTRO - COUNTER 0, USART 0  
ITI\$CNTRI - COUNTER 1, USART 1

IT1\$CNTR2 - COUNTER 2, USART 2  
IT2\$CONT - INTERVAL TIMER 2 CONTROL  
IT2\$CNTR0 - COUNTER 3, USART 3  
IT2\$CNTR1 - COUNTER 4, CNTR5 OR SPLIT CLOCKS  
IT2\$CNTR2 - COUNTER 5, RST 7.5  
USART\$CNTR\$M3 - DIVIDE BY N RATE GENERATOR, MODE 3, FOR USART CLK \*  
16, CLK = 1.2288 MHZ  
B19200 - TIMER VALUE FOR 19.2 KBPS  
B9600 - TIMER VALUE FOR 9600 BPS  
B4800 - TIMER VALUE FOR 4800 BPS  
B2400 - TIMER VALUE FOR 2400 BPS  
B1200 - TIMER VALUE FOR 1200 BPS  
B600 - TIMER VALUE FOR 600 BPS  
B300 - TIMER VALUE FOR 300 BPS  
B150 - TIMER VALUE FOR 150 BPS  
B110 - TIMER VALUE FOR 110 BPS

8155 Peripheral Interface Constants:

PI\$PORTA - PORT A (OUTPUT)  
PI\$PORTB - PORT B (INPUT)  
PI\$PORTC - PORT C (INPUT)  
PI\$STAT - PPI STATUS  
PI\$CMD - PPI COMMAND  
PI\$CNTR\$LO - PPI COUNTER LC BYTE  
PI\$CNTR\$HI - PPI COUNTER HI BYTE  
PI\$CNTR\$LOCNT - PPI COUNTER TIME CONST  
PI\$CNTR\$HICNT - PPI COUNTER TIME CONST  
PI\$INIT\$CMD1 - PPI INITIALIZATION COMMAND 1, A OUT, B & C IN, STOP  
COUNT  
PI\$INIT\$CMD2 - PPI INITIALIZATION COMMAND 2, A OUT, B & C IN, START  
COUNT  
PI\$INIT\$US\$INT1 - USART AND INT CONT RESET  
PI\$INIT\$US\$INT2 - USART AND INT CONT NORMAL  
PI\$PORTC\$STAT - PORT C STATUS  
PI\$PORTC\$CTL - PORT C CONTROL  
PI\$M2M1 - A-MODE 1, B-MODE 2  
PI\$OBF - OUTPUT BUFFER READY  
PI\$IBF - INPUT BUFFER READY

8259 Interrupt Controller Constants:

IC\$PORTA - PORT A  
IC\$PORTB - PORT B  
IC\$ICW1 - INIT COMMAND WORD 1, (A7A6A5) = 010; EDGE TRIG; INTERVAL  
= 4; SINGLE; NO ICW4  
IC\$ICW2 - INIT COMMAND WORD 2, (A15-A0) = 0  
IC\$ICW3 - INIT COMMAND WORD 3, NO SLAVE IN IR  
INIT\$MASK - '10101010B', INITIAL INTERRUPT MASK, OCW1; RECEIVE INTR  
ON, TRANSMIT INTR OFF  
IC\$EOI - END OF INTERRUPT CMD, OCW2, ROTATE (PRIORITY) ON NON-  
SPECIFIC EOI  
IC\$OCW3\$SMMS - SPECIAL MASK MODE SET  
IC\$OCW3\$SMR - SPECIAL MASK MODE RESET

Variables

BYTES\$RECV\$1, BYTES\$RECV\$2, BYTES\$RECV\$3, BYTES\$RECV\$4 - Integer value  
indicating how many bytes of a datagram have been received from a  
host.

BYTES\$SENT\$1, BYTES\$SENT\$2, BYTES\$SENT\$3, BYTES\$SENT\$4 - Integer value  
indicating how many bytes of a datagram have been sent to the host.

CHAR\$1, CHAR\$2, CHAR\$3, CHAR\$4 - Place holder for the received character  
in the receive interrupt routine.

RXTA\$1, RXTA\$2, RXTA\$3, RXTA\$4 - Boolean flag to indicate if a transmit  
acknowledge has been received.

RXTR\$1, RXTR\$2, RXTR\$3, RXTR\$4 - Boolean flag to indicate if a transmit  
request has been received.

SEND\$1, SEND\$2, SEND\$3, SEND\$4 - Boolean flag to indicate when a host  
channel is sending data to its host.

TA - Transmit acknowledge character.

TR - Transmit request character.

TRTA\$1, TRTA\$2, TRTA\$3, TRTA\$4 - Boolean flags to indicate if the  
transmit request/transmit acknowledge handshake is in use.

TXTA\$1, TXTA\$2, TXTA\$3, TXTA\$4 - Boolean flag to indicate if a transmit  
acknowledge was sent.

TXTR\$1, TXTR\$2, TXTR\$3, TXTR\$4 - Boolean flag to indicate if a transmit  
request was sent.



Procedures

INITIALIZE\$BOARD - Initialize the hardware integrated circuits on the SBC 544.

R\$MASK - External procedure to read the interrupt mask on the 8085 processor. Linked from PLM80.LIB.

S\$MASK - External procedure to set the interrupt mask on the 8085 processor. Linked from PLM80.LIB.

Link and Locate Batch File

CAUTION: Do not change address or other parameters in the following batch file. They are highly 544 hardware dependent on the SBC 544 architecture and the network layer software.

```
LINK OP544.OBJ,PLM80.LIB TO OP544.LNK MAP
LOCATE OP544.LNK TO OP544 STACKSIZE(100H) ORDER(CODE,DATA,&
STACK,MEMORY) CODE(60H) DATA(0A000H)&
RESTARTO MAP PRINT(OP544.MP2)
TYPE OP544.MP2
OBJHEX OP544 TO OP544.HEX
```

#### 4. ISIS Host Simulation (15:Appendix G)

The purpose of this program is to simulate a host system to the SBC 544 network layer software. All the constants, variables and procedures from the network layer simulation are used in this program with the exception that the console I/O now occurs through the either CP/M or ISIS system calls and the actual character I/O to the UNID II is accomplished through calls to an I/O module linked to a this module.

The following constants, variables and procedures are additions to the network layer simulation.

##### Constants

ASCII(\*) - Array used for converting hex to binary and binary to hex.  
BDOS2 - BDOS call 2-console output (not used with ISHOST).  
BDOS9 - BDOS call 9-print string until °\$(not used with ISHOST).  
BDOS10 - BDOS CALL 10-read console input buffer (not used with ISHOST).  
DATA\$GRAM\$SIZE - Number of bytes from host.  
DATA\$TABLE\$SIZE - Number of bytes in datagram table.  
MAX\$COUNTRY\$CODE - Indicates country codes in use.  
MAX\$NETWORK\$CODE - Indicates UNIDs operational in the network.  
MAX\$RX\$TRIES - Maximum number of TA wait tries.  
PACKET\$TABLE\$SIZE - Number of bytes in packet table.  
TCP\$DATA\$SIZE - TCP data size.  
THIS\$COUNTRY\$CODE - Country code where this UNID resides.  
THIS\$UNID\$NBR - Unique address for this UNID in its country code.

## Variables

RESULT - Error value returned by BDOS function calls.

BUFFER(128) - Line buffer used for console input.

CHAN\$NUM - Channel number in which to load the test datagrams.

DEST\$NET\$CODE - Destination network code for the test datagrams.

DEST\$HOST\$CODE - Destination host code for the test datagrams.

CHAN\$PTR - Pointer to the current datagram.

RXTA\$TRIES - Number of received transmit acknowledge attempts.

TRANS\$RDY - Indicator when the transmit software is ready to transmit a datagram.

RX01NE - Pointer to next available space to receive a datagram.

RX01NS - Pointer to next datagram to service.

RX01SZ - Size of receive datagram buffer.

RX01TB - Receive buffer for datagrams.

TX01NE - Pointer to next available space to send a datagram.

TX01NS - Pointer to next datagram to service

TX01SZ - Size of receive datagram buffer.

TX01TB - Transmit buffer for datagrams.

DESTINATION - Destination of the datagram for program control.

DESTINATION\$ADDRESS - Destination address of datagram from IP header.

SOURCE\$ADDRESS - Source address of datagram from IP header.

BYTES\$RECV - Integer value indicating how many bytes of a datagram have been received from a host.

BYTES\$SENT - Integer value indicating how many bytes of a datagram have been sent to the host.

CHAR - Place holder for the received character in the receive interrupt routine.

RXTA - Boolean flag to indicate if a transmit acknowledge has been received.

RXTR - Boolean flag to indicate if a transmit request has been received.  
SEND - Boolean flag to indicate when a host channel is sending data to its host.  
TA - Transmit acknowledge character.  
TR - Transmit request character.  
TRTA - Boolean flags to indicate if the transmit request/transmit acknowledge handshake is in use.  
TXTA - Boolean flag to indicate if a transmit acknowledge was sent.  
TXTR - Boolean flag to indicate if a transmit request was sent.

#### Procedures

BDOS - External call to the CP/M operating system to perform a BDOS call (not used with ISHOST).  
CHK\$RXTA - Procedure to check the receive USART for a received transmit acknowledge character.  
CHK\$RXTR - Procedure to check the receive USART for a received transmit request character.  
EXIT - External call to return to the CP/M or ISIS operating system.  
INIT - Procedure to initialize the variables used in the program.  
LD\$TAB\$H\$SKP - Procedure to adjust the pointers to the next available datagram position in a buffer table.  
LOAD - Procedure to interactively load datagrams into a buffer for transmission to the UNID.  
LOOP2 - Procedure to send and receive a datagram.  
RCV\$1 - Procedure to read a datagram from the USART.  
READ - Procedure to read a line of buffered input from the host console.  
READ\$LINE - Procedure to interactively read and interpret a line of text at the host console.  
READ\$RXTAB - Procedure to read and display the contents of the receive buffer table.  
READ\$TXTAB - Procedure to read and display the contents of the transmit buffer table.

SCLRCM - External call to clear the USART receive port.  
SCMCHK - External call to check the USART receive port for a character.  
SCMIN - External call to get a character from the USART.  
SCMOUT - External call to send a character to the USART.  
SINIT - External call to initialize the host USART port.  
SNDSEQ - Procedure to send a message to the host console for display.  
SRVC\$TAB\$H\$SKP - Procedure to adjust the pointers to the next to service datagram in the buffer tables.  
TRANS\$1 - Procedure to send a datagram to the USART.

#### MODULE INFORMATION

MODULE HOST1.ASM - for use with the original CPMTMP.SRC code used on a CP/M machine using an 8251/8251A USART (i.e. Intel 210 under CP/M).  
MODULE HOST2.ASM - for use with ISIS operating system or any operating system to which the host software has been transported. The I/O ports use 8251/8251A USARTs.  
MODULE HOST3.SRC - same as HOST2.ASM, except the source code is written in PL/M.

#### Link and Locate Batch File

CAUTION: Do not change address or other parameters in the following batch file. They are highly CP/M system dependent.

For linking ISHOST with HOST3.SRC:

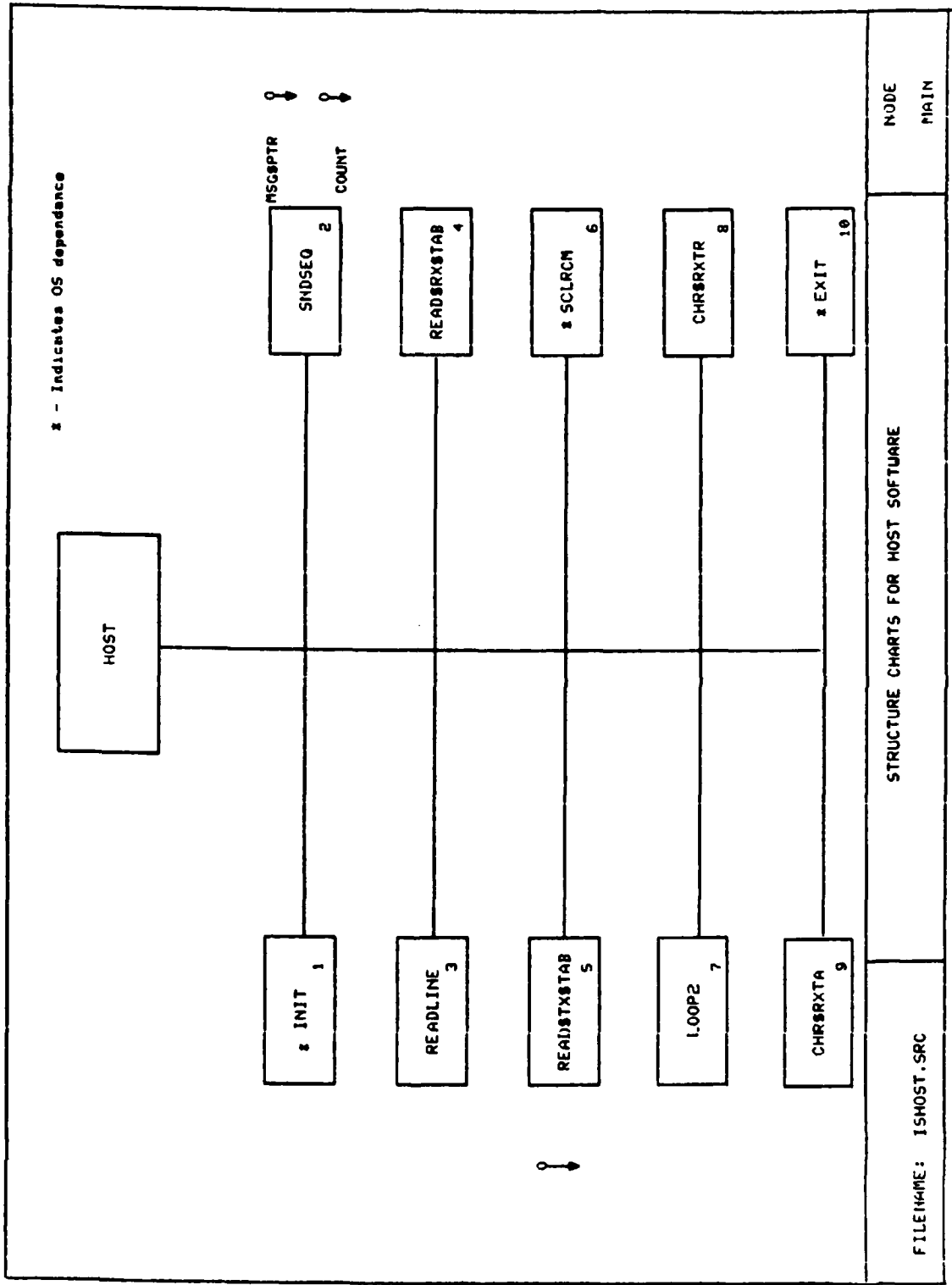
```
LINK ISHOST.OBJ, HOST3.OBJ, PLM80.LIB TO ISHOST.LNK MAP
LOCATE ISHOST.LNK TO ISHOST STACKSIZE(100H) ORDER(CODE,DATA,&
STACK,MEMORY) CODE(103H) MAP PRINT(ISHOST.MP2)
TYPE ISHOST.MP2
OBJHEX ISHOST TO ISHOST.HEX
```

For installing the the linked and located code on the SBC544 EPROM, refer to Appendix D.

APPENDIX I

UNID II Software Structure charts

1. ISIS HOST: ISHOST.SRC - Programs the Intel 230 to act as a test host for the UNID II. . . . . I-2
2. OPERAITONAL SBC 544: OP544.SRC - Program resident on the SBC 544 . . . . . I-7
3. DATA LINK SIMULATION: NETX25.SRC, LAPB0.SRC, LAPB1.SRC, PCKT.SRC. . . . . I-15



INIT  
1.0

\* SINIT  
1.1

\* SCLROM  
1.2

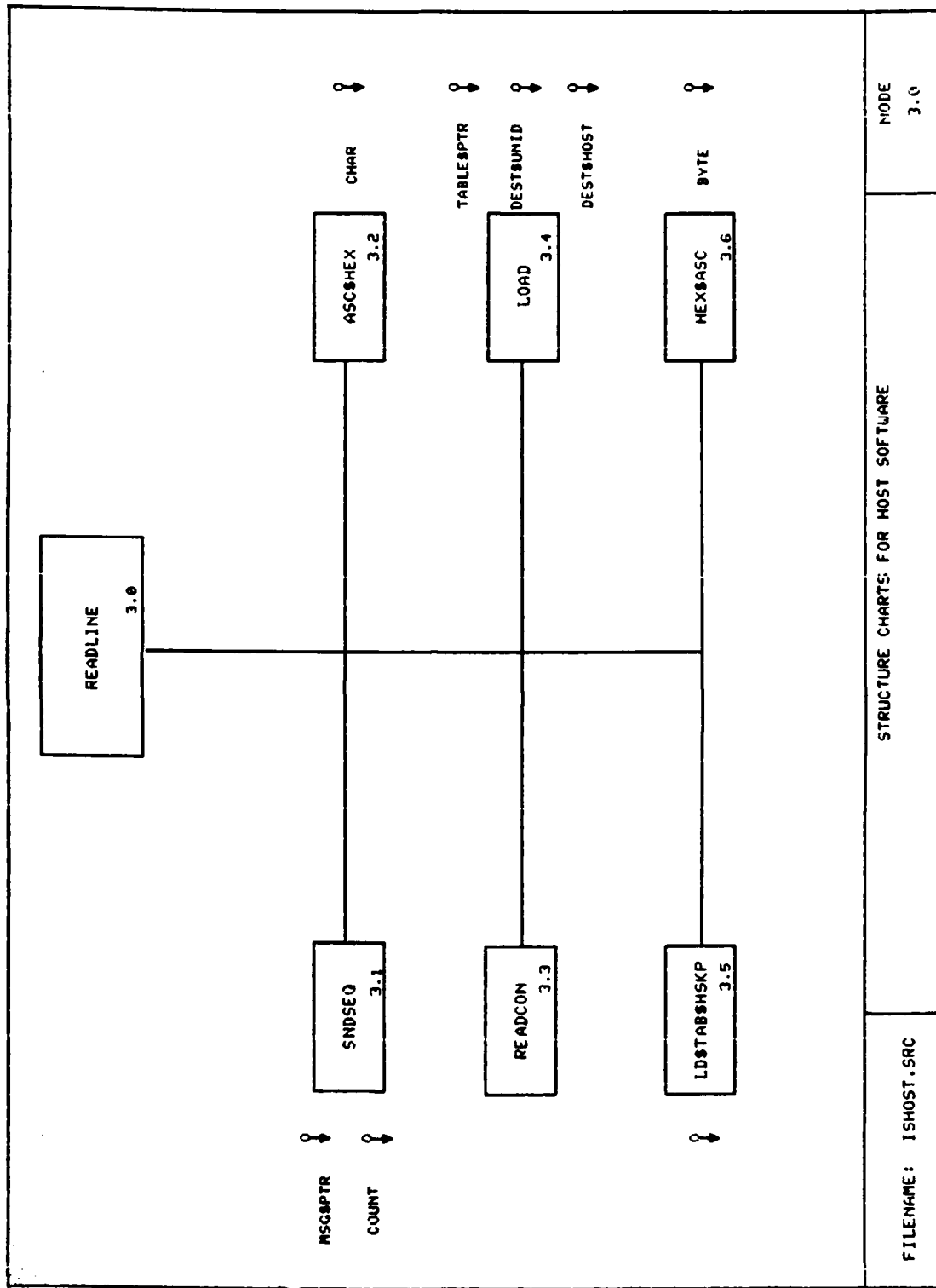
\* - INDICATES HAREWARE DEPENDENT PROCEDURES

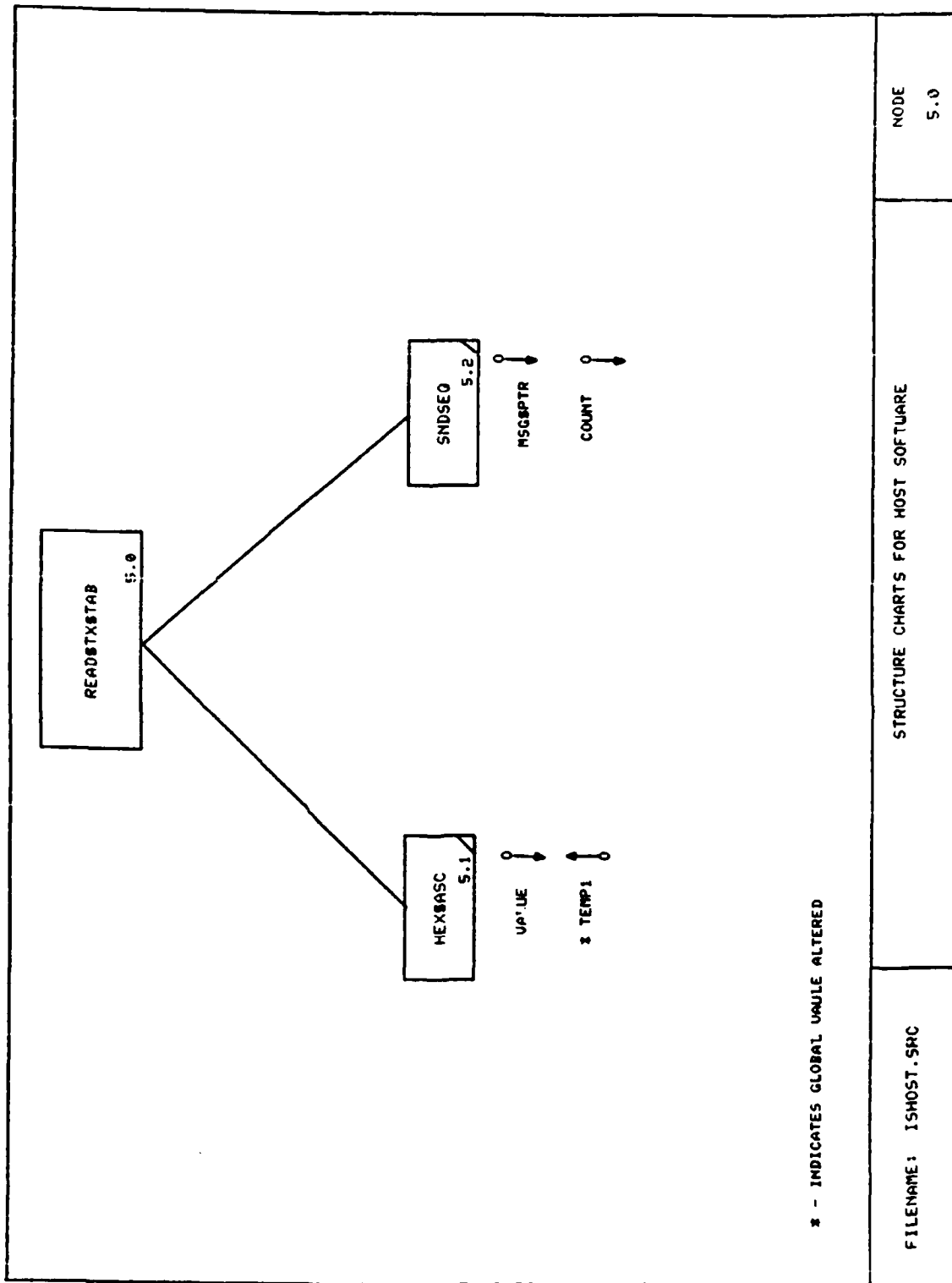
FILENAME: ISHOST.SRC

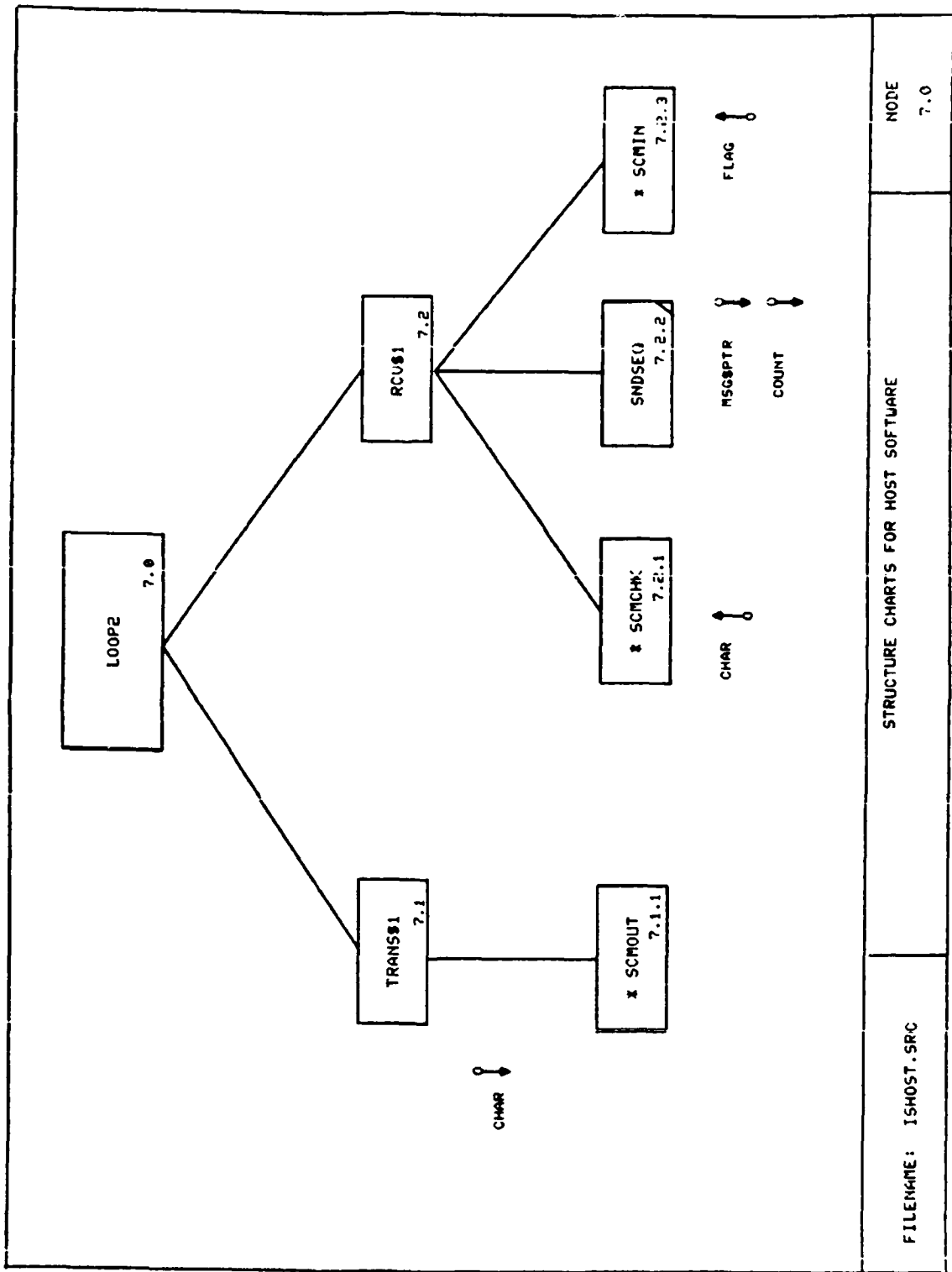
STRUCTURE CHARTS FOR HOST SOFTWARE

MODE  
1.0

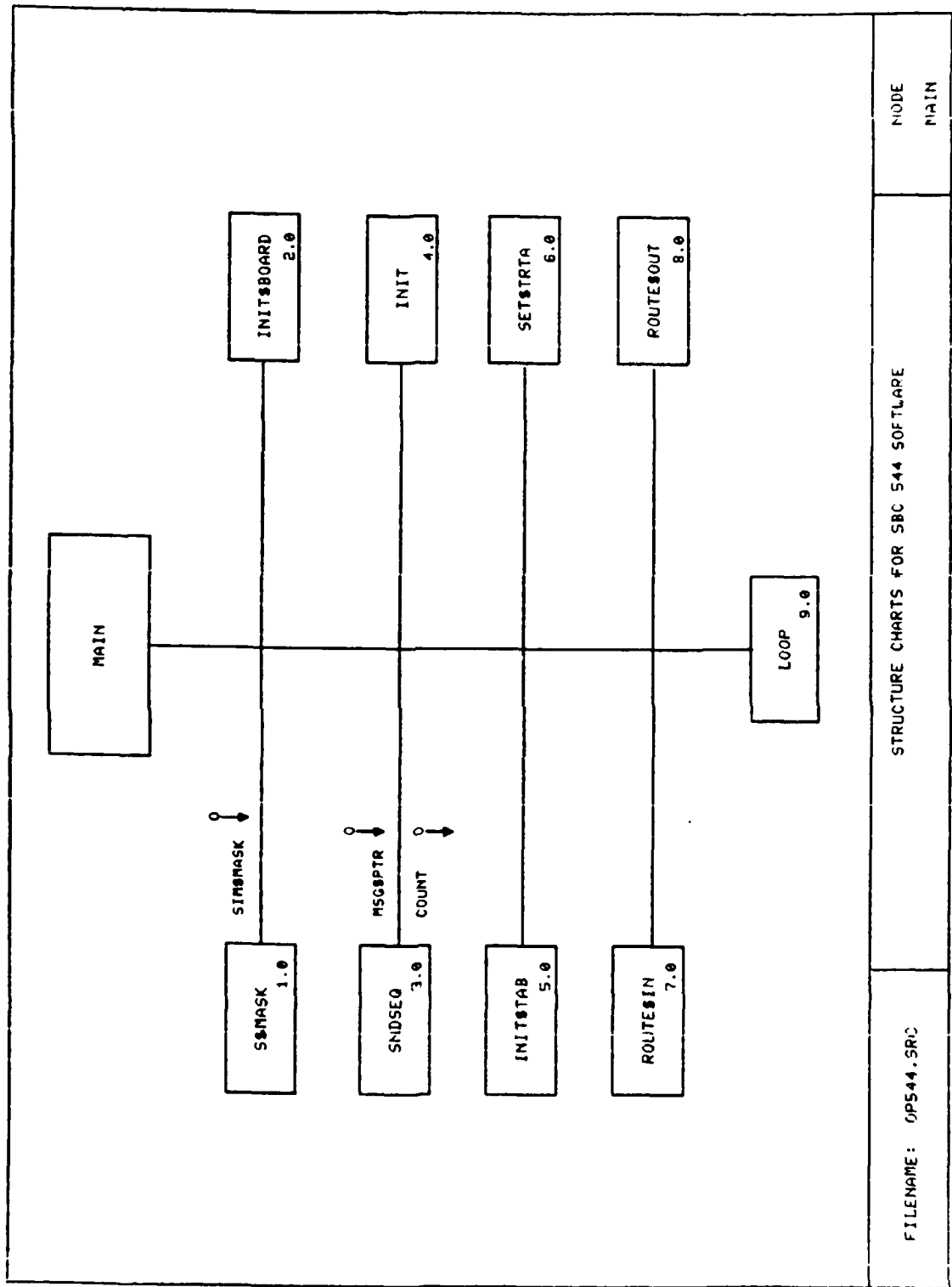




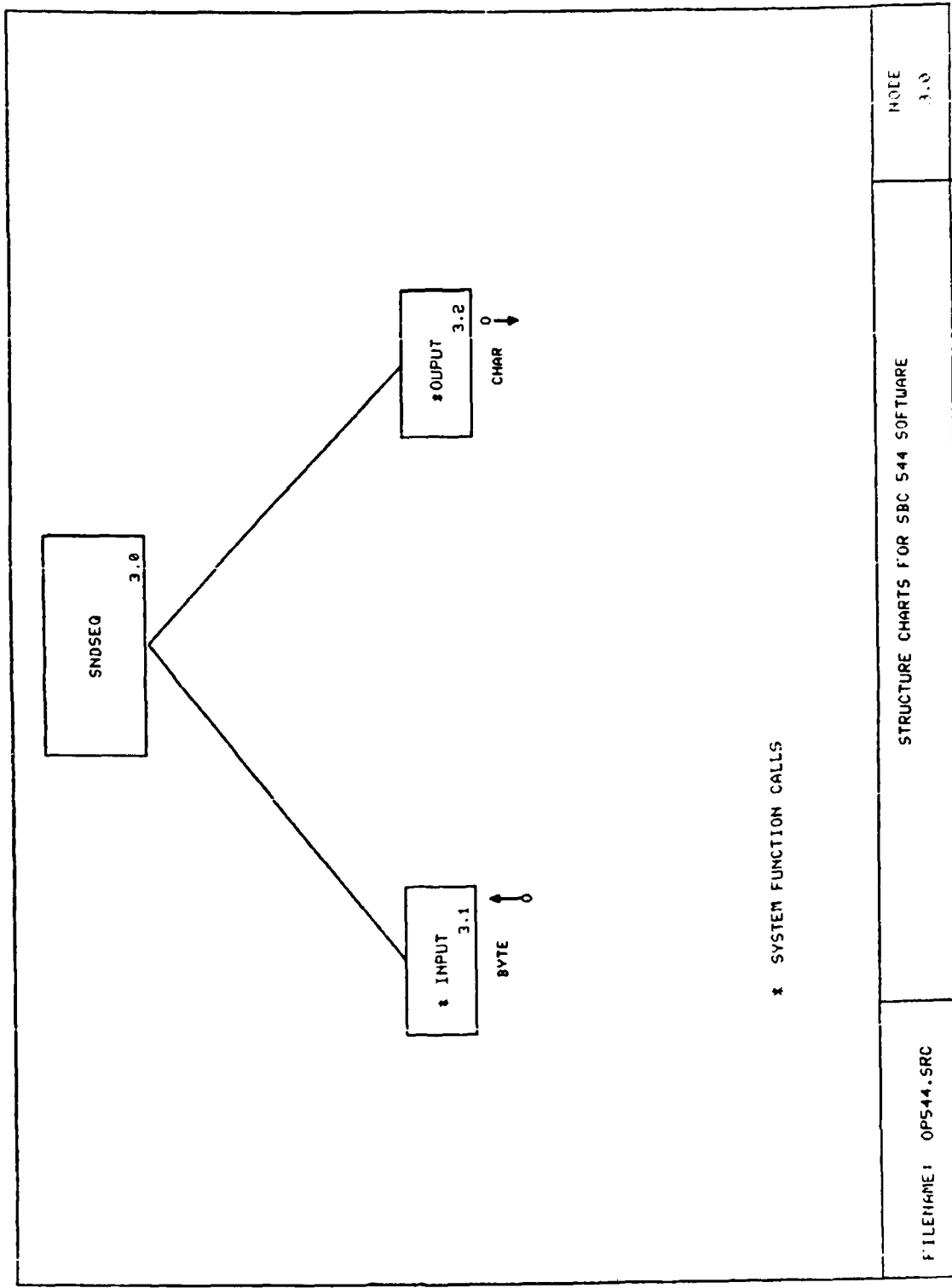




2. OPERAITONAL SBC 544: OP544.SRC - Program resident on  
the SBC 544

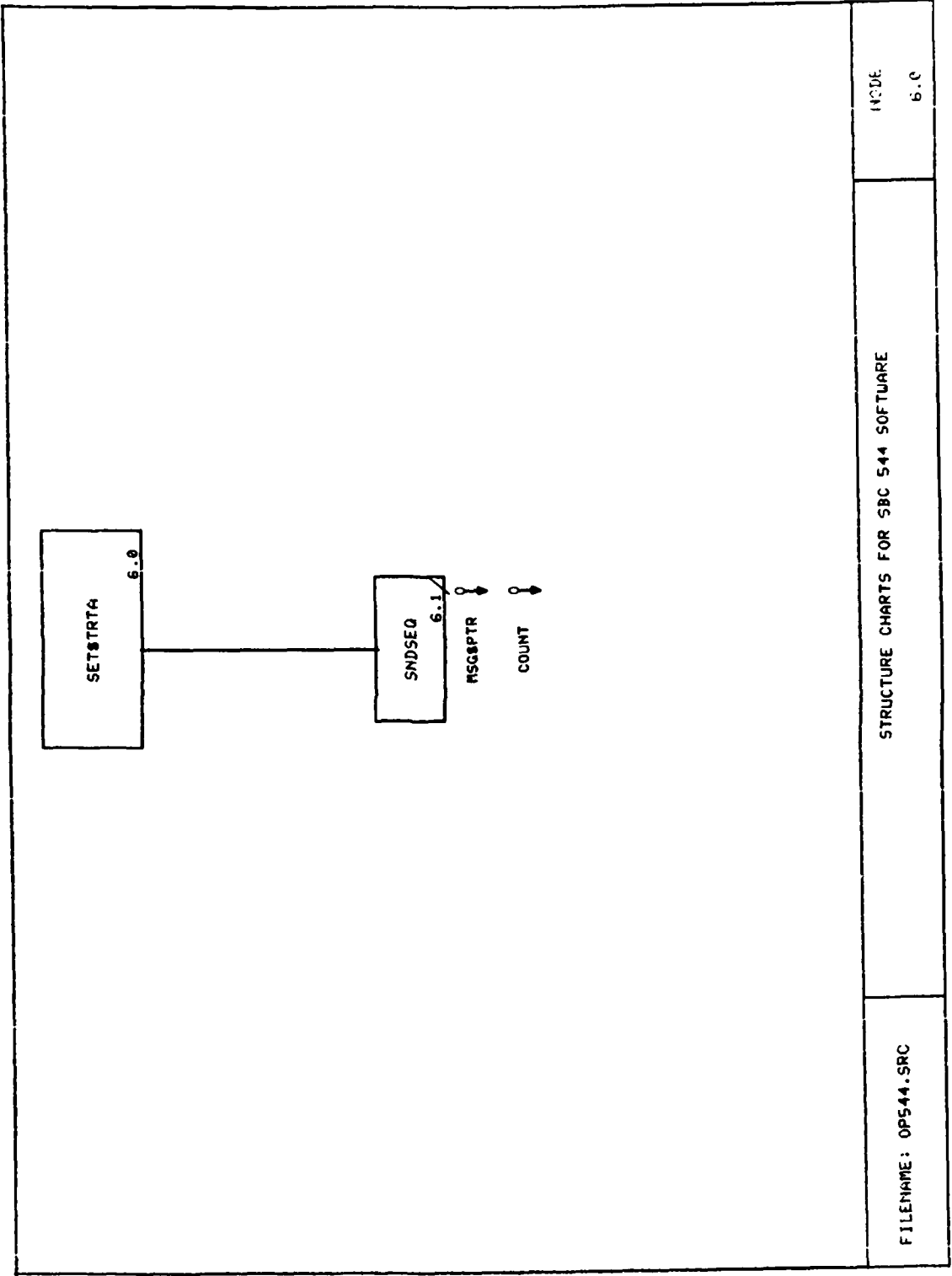


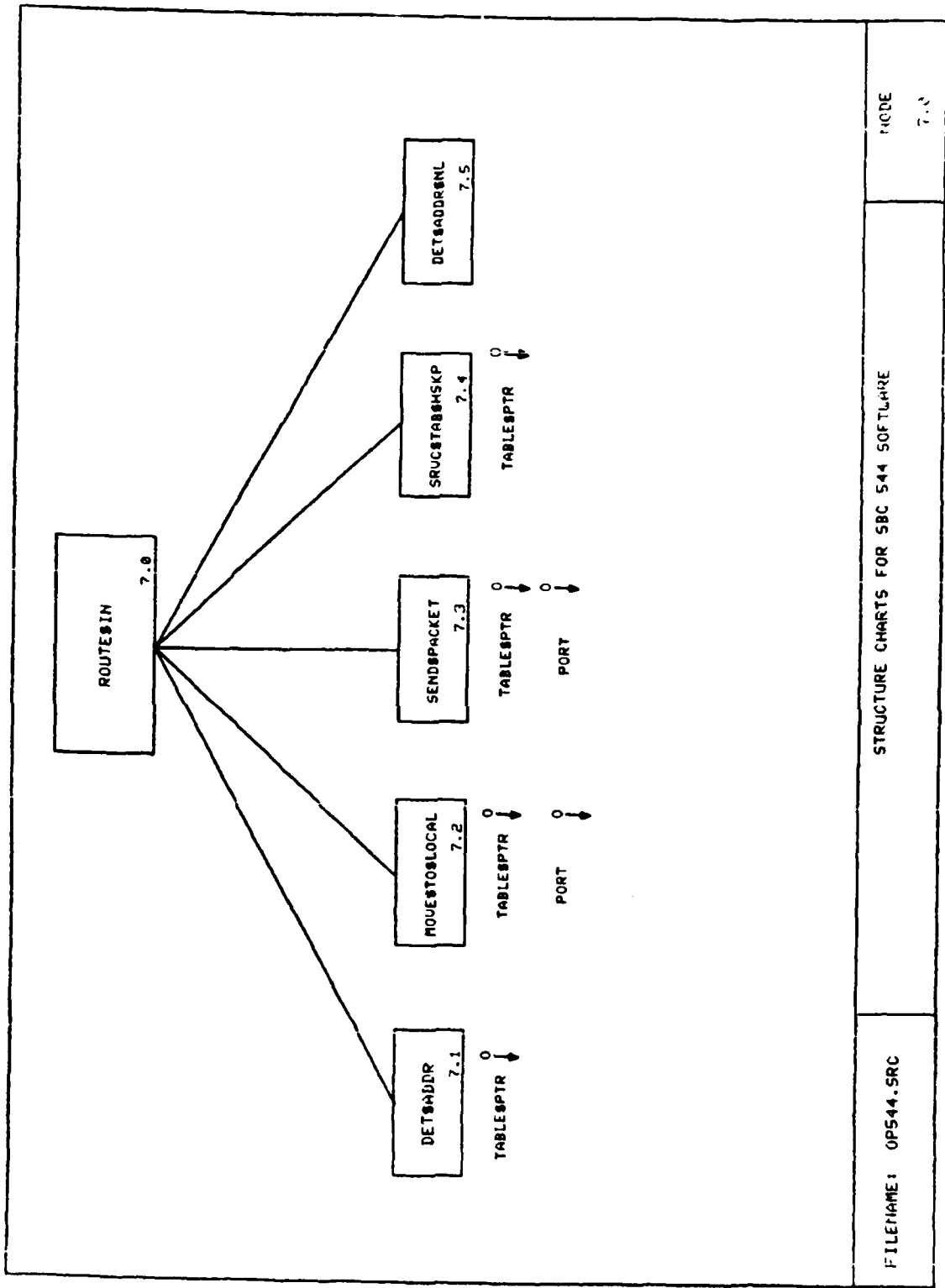
FILENAME: OP544.SRC	STRUCTURE CHARTS FOR SBC 544 SOFTWARE	NODE MAIN
---------------------	---------------------------------------	--------------



\* SYSTEM FUNCTION CALLS

FILENAME: OP544.SRC	STRUCTURE CHARTS FOR SBC 544 SOFTWARE	NOTE 3.0
---------------------	---------------------------------------	-------------





STRUCTURE CHARTS FOR SBC 544 SOFTWARE

MODE 7.0

FILENAME: OPS44.SRC



MOVESTOBLOCAL  
7.2

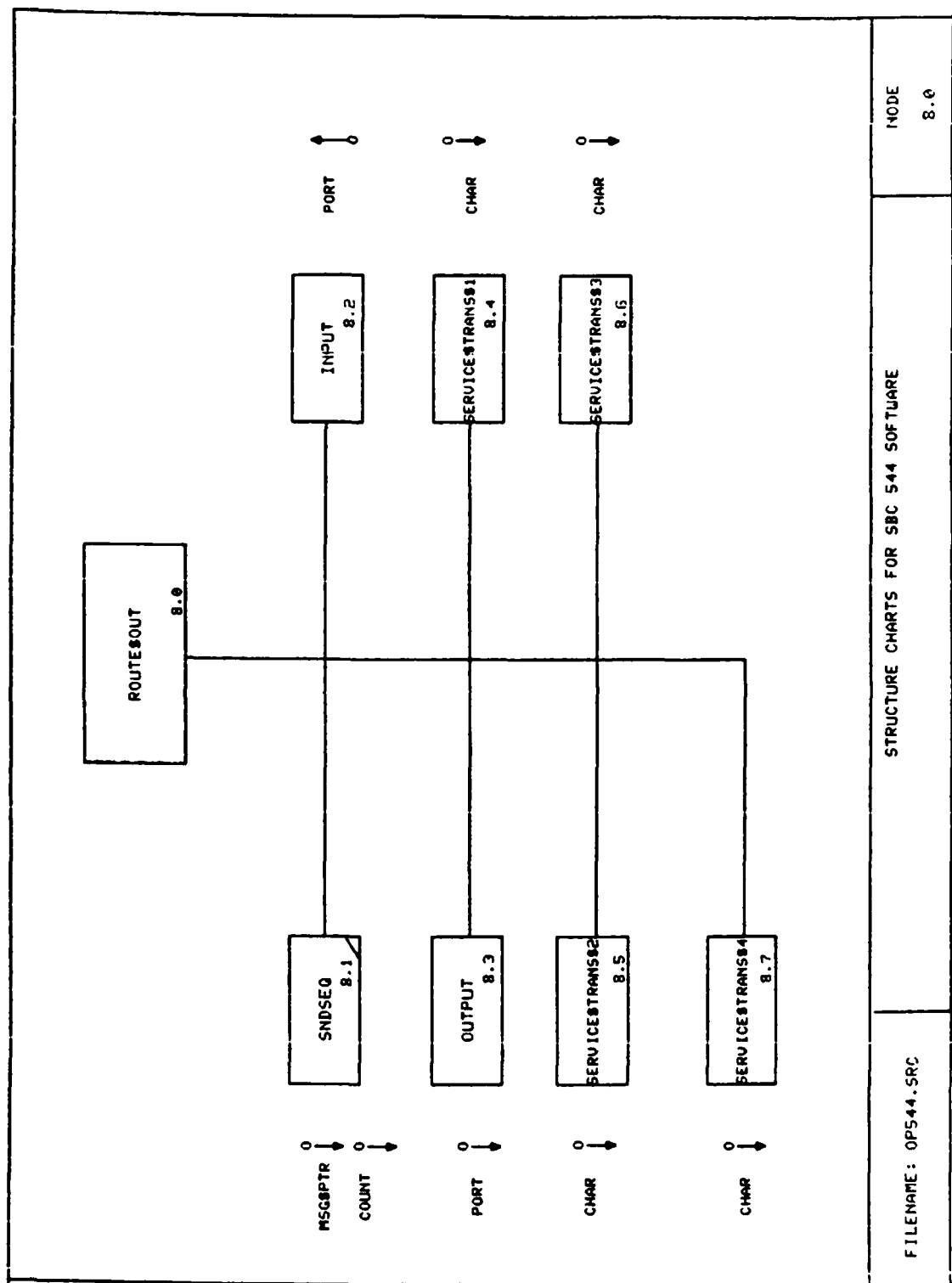
MOVE  
7.2.1

SIZE →  
TABLESPTR →

LDSTABMSKP  
7.2.2

TABLESPTR →

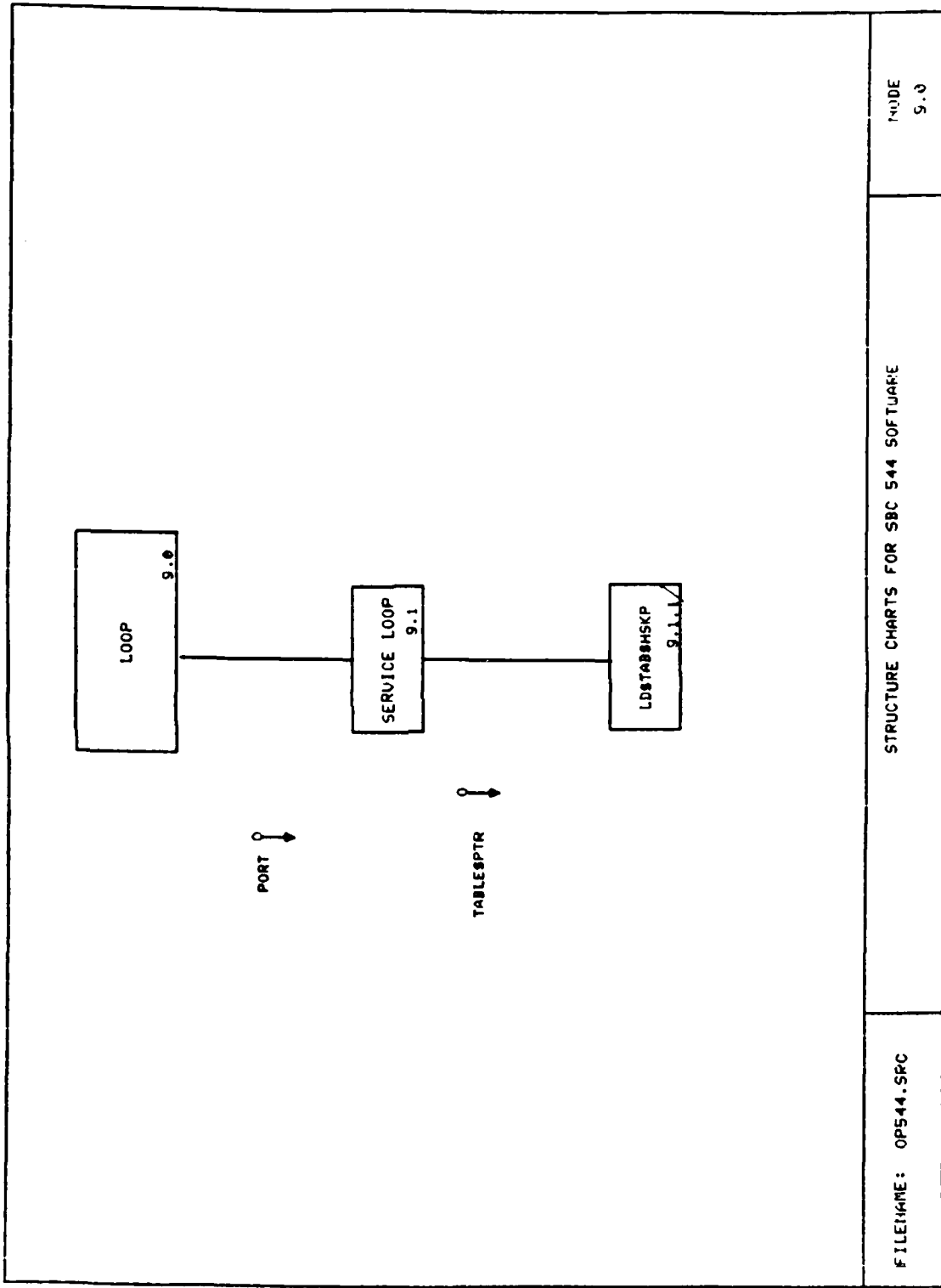
FILENAME: OPS44.SPC	STRUCTURE CHARTS FOR SBC 544 SOFTWARE	NOJL 7.2
---------------------	---------------------------------------	-------------



STRUCTURE CHARTS FOR SBC 544 SOFTWARE

MODE  
8.0

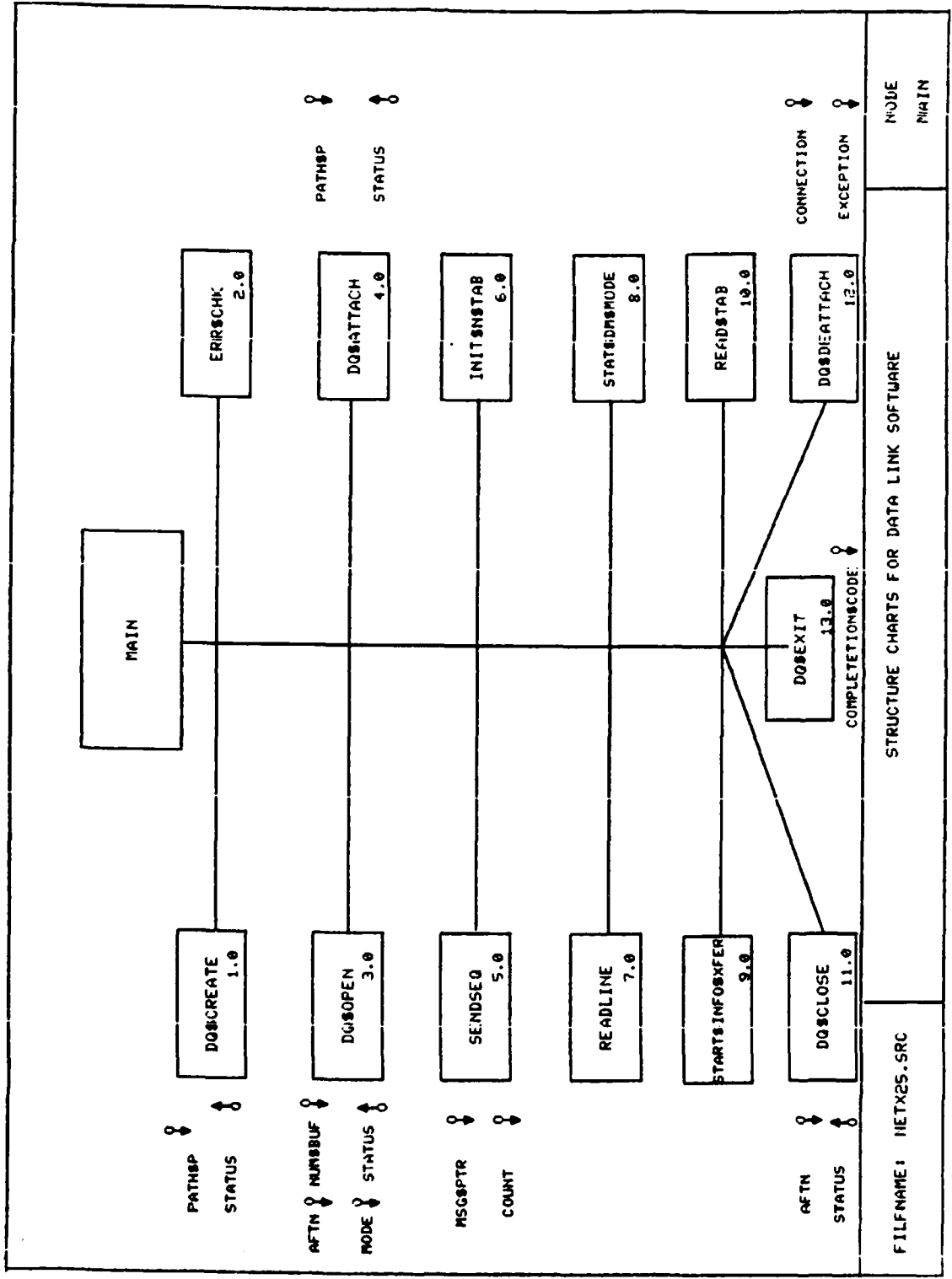
FILENAME: OP544.SRC



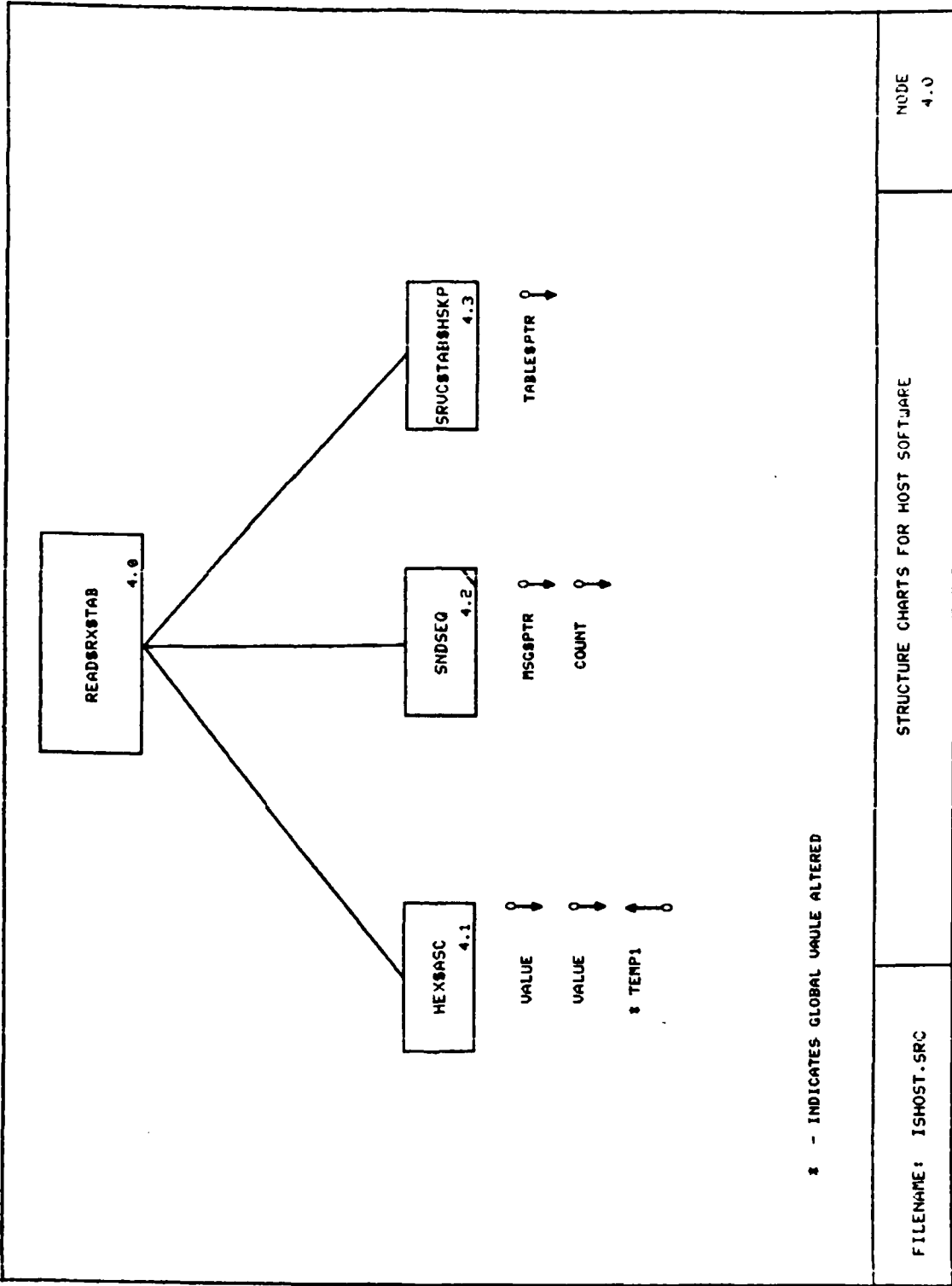
MODULE  
9.0

STRUCTURE CHARTS FOR SBC 544 SOFTWARE

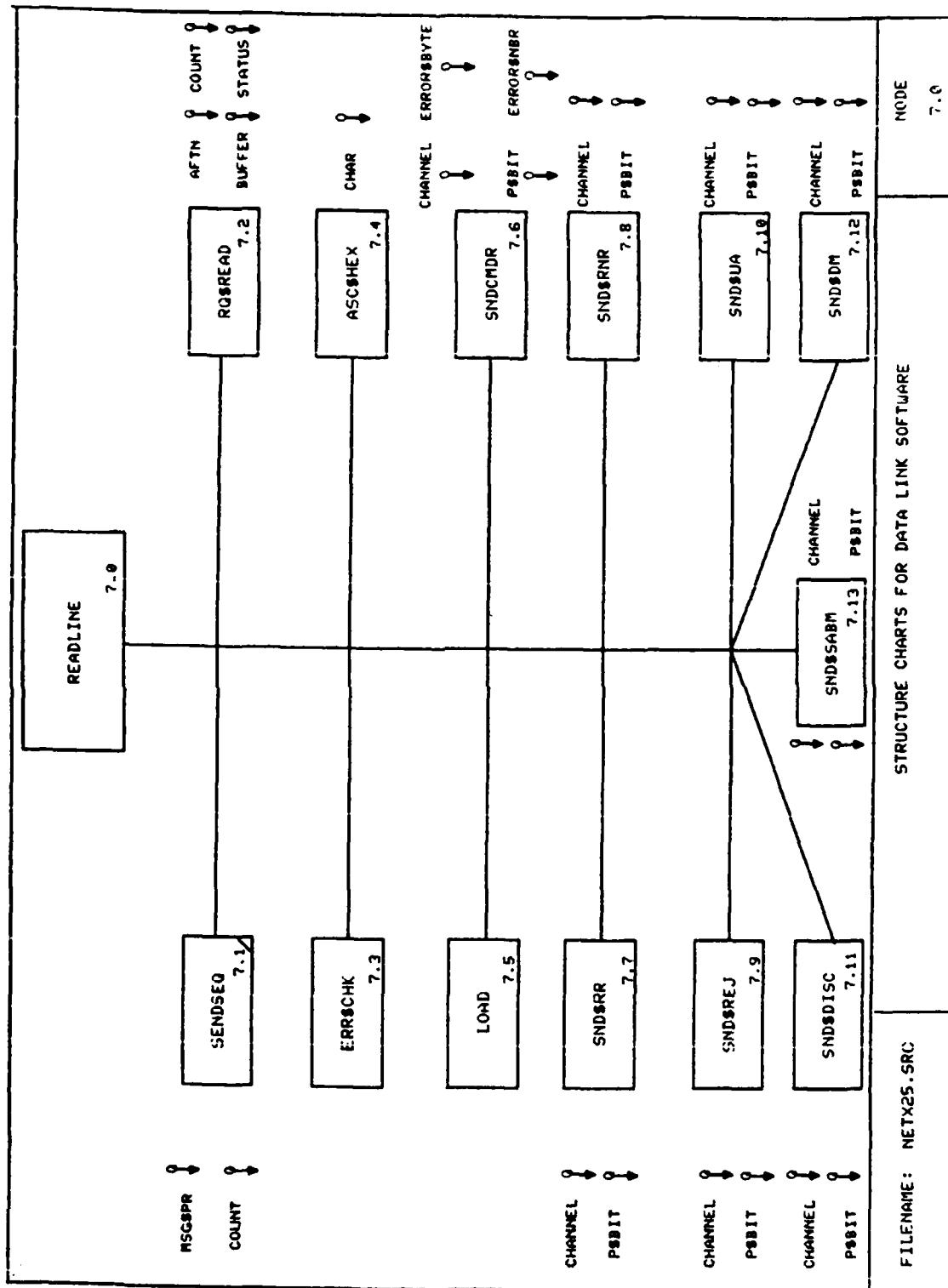
FILENAME: OP544.SRC

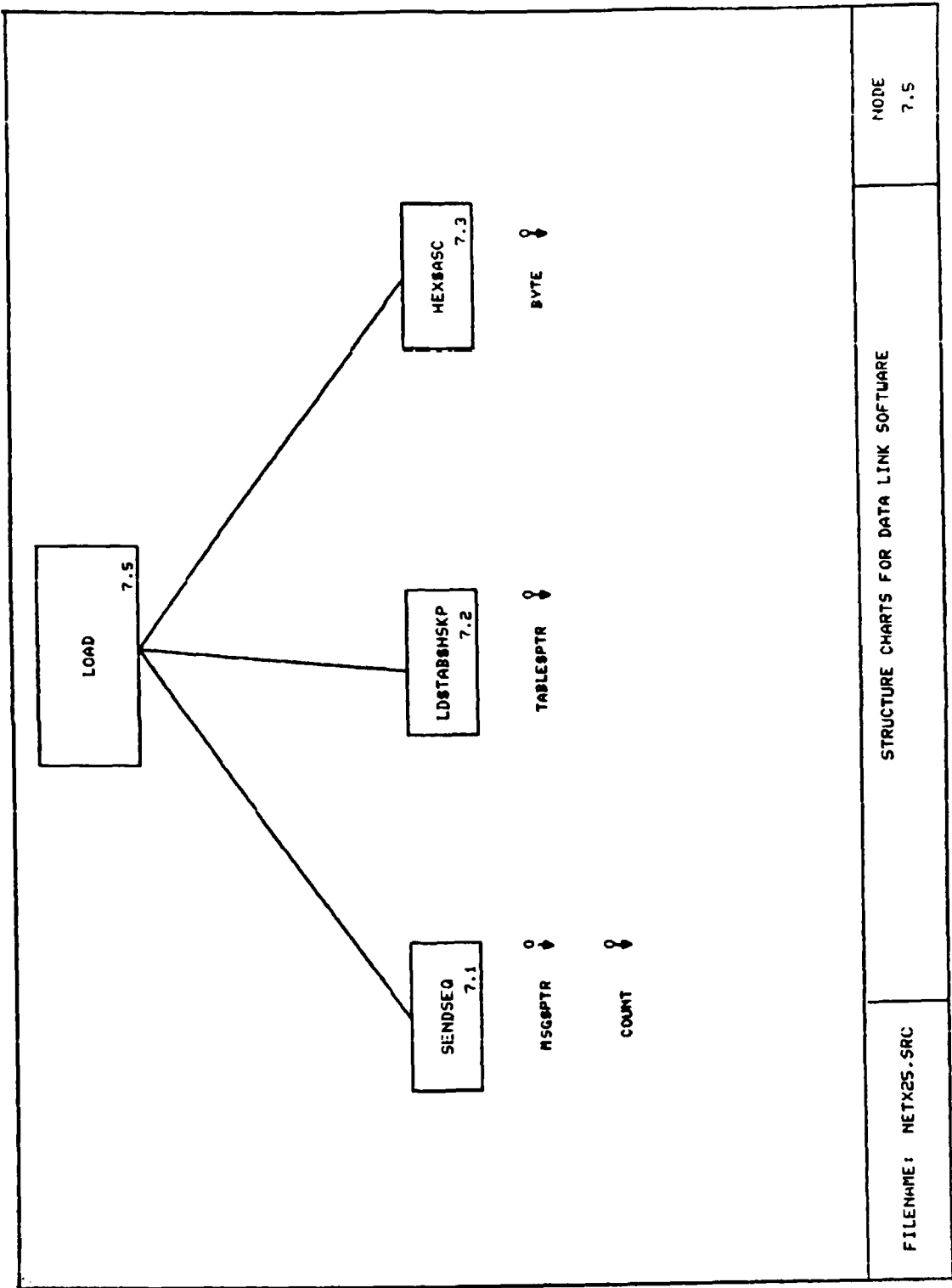


3. DATA LINK SIMULATION: NETX25.SRC, LAPBO.SRC, LAPBI.SRC,  
PCKT.SRC.

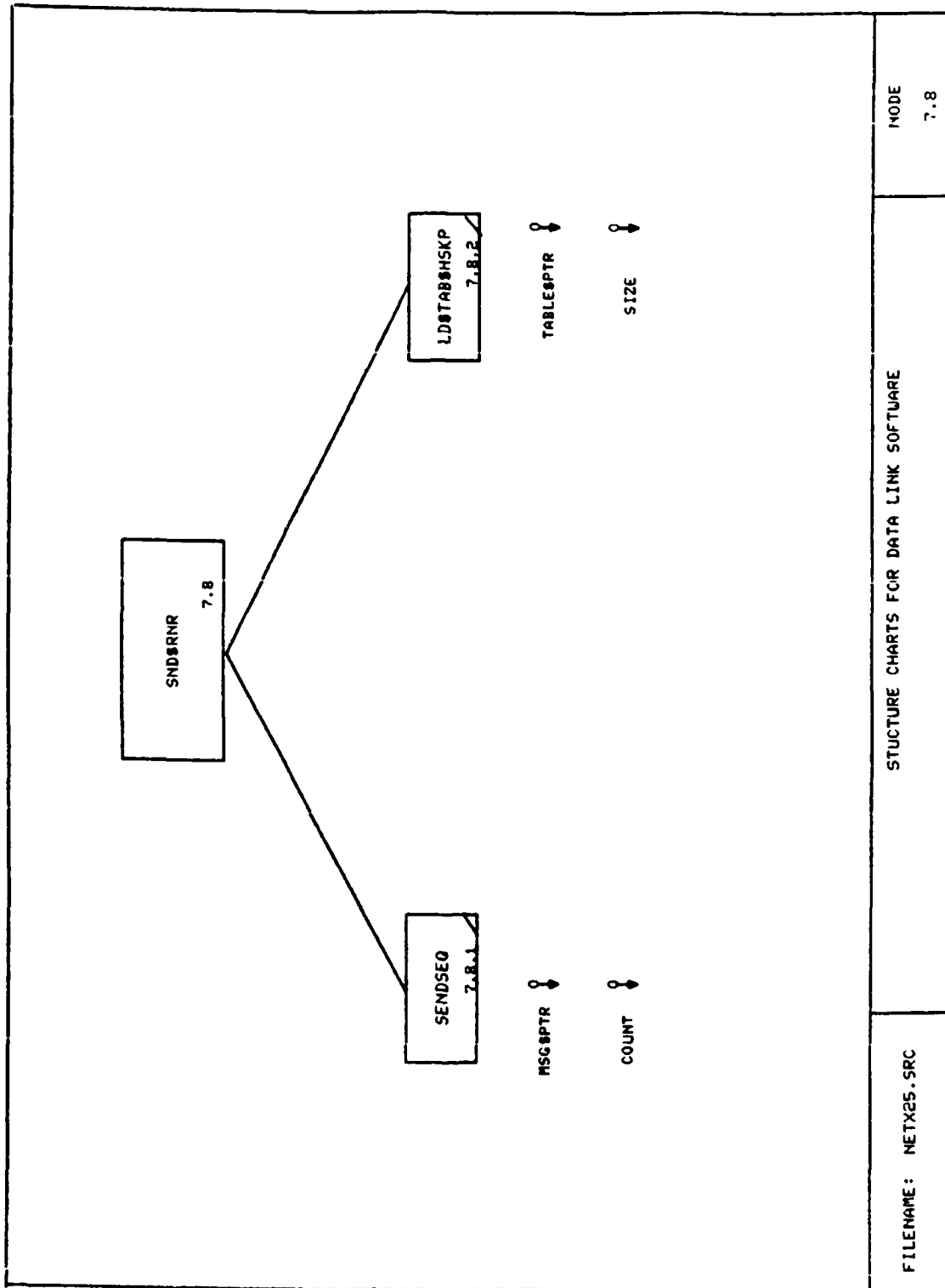


FILENAME: ISHOST.SRC	STRUCTURE CHARTS FOR HOST SOFTWARE	NODE 4.0
----------------------	------------------------------------	-------------





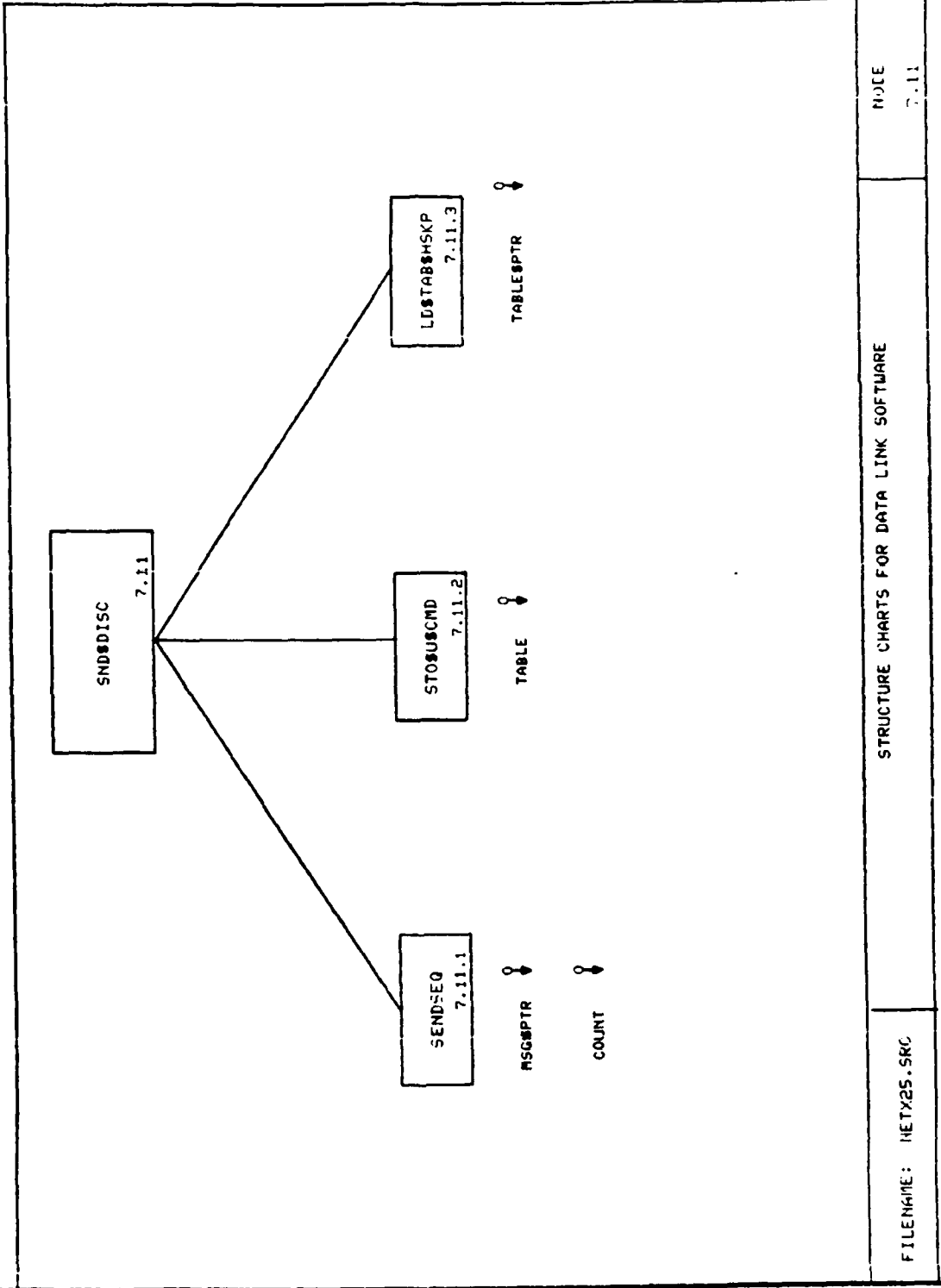




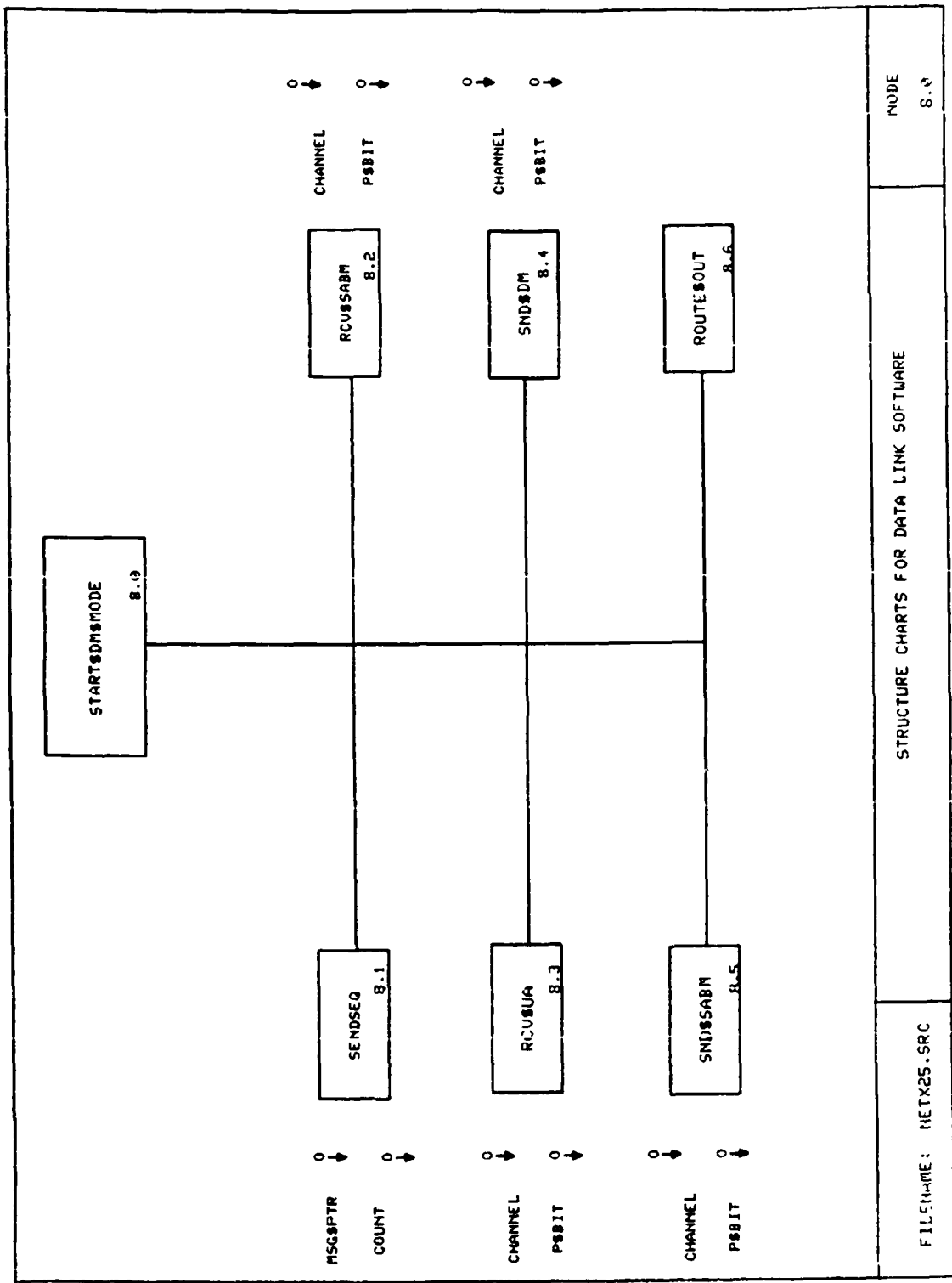
STRUCTURE CHARTS FOR DATA LINK SOFTWARE

FILENAME: NETX25.SRC

MODE  
7.8



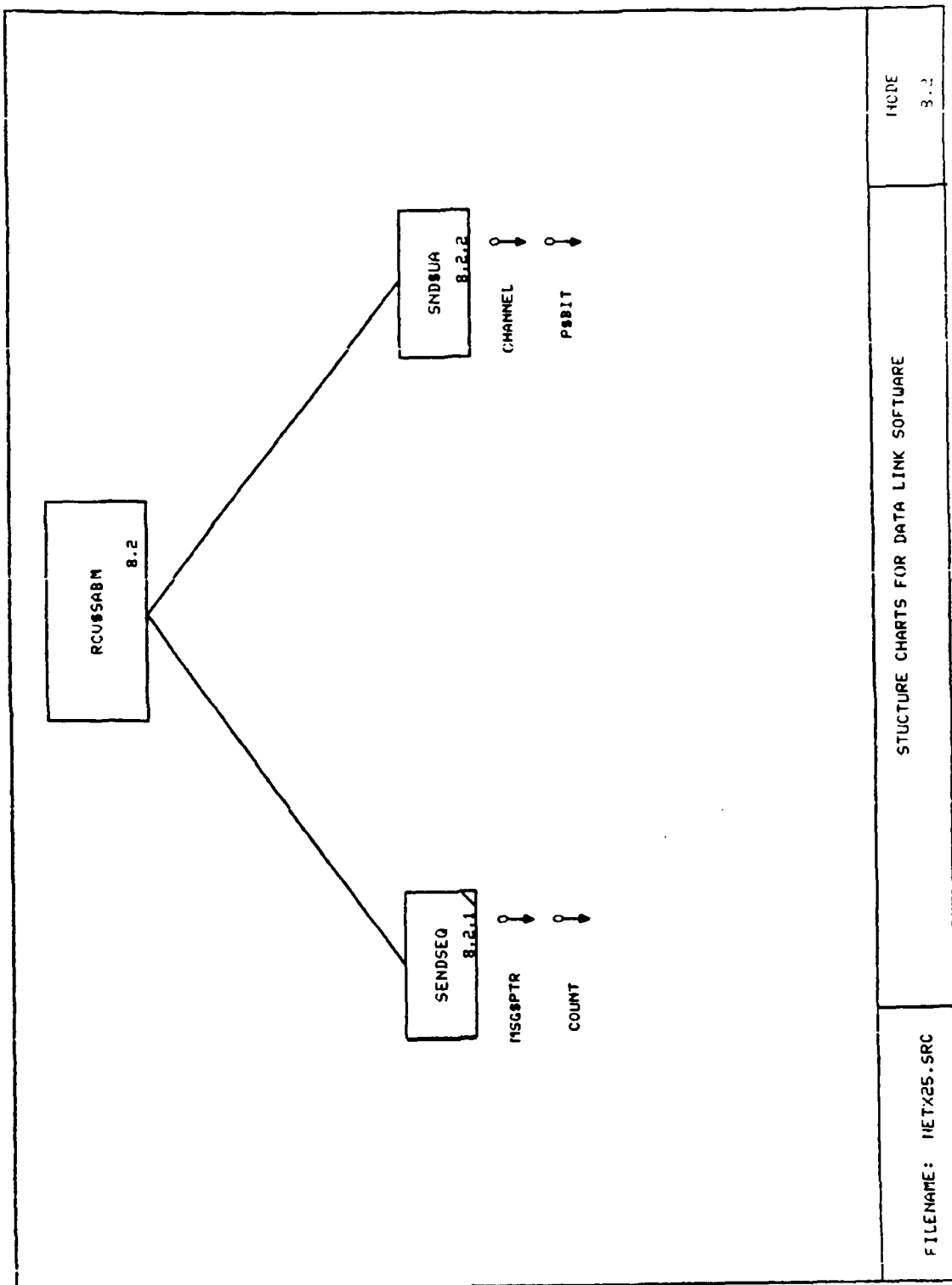
FILENAME: NETX25.SRC	STRUCTURE CHARTS FOR DATA LINK SOFTWARE	NODE 7.11
----------------------	---	--------------



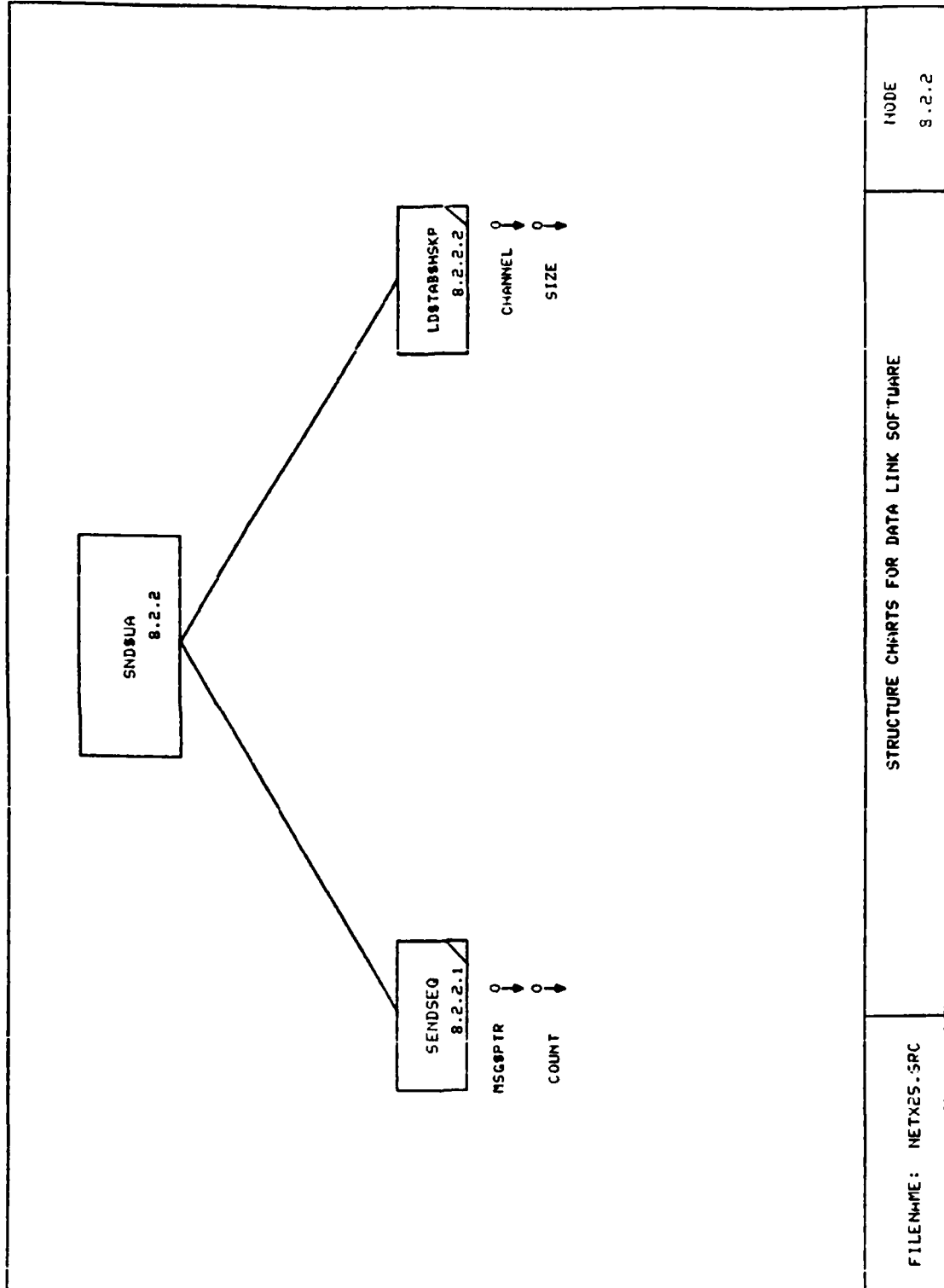
STRUCTURE CHARTS FOR DATA LINK SOFTWARE

FILENAME: NETX25.SRC

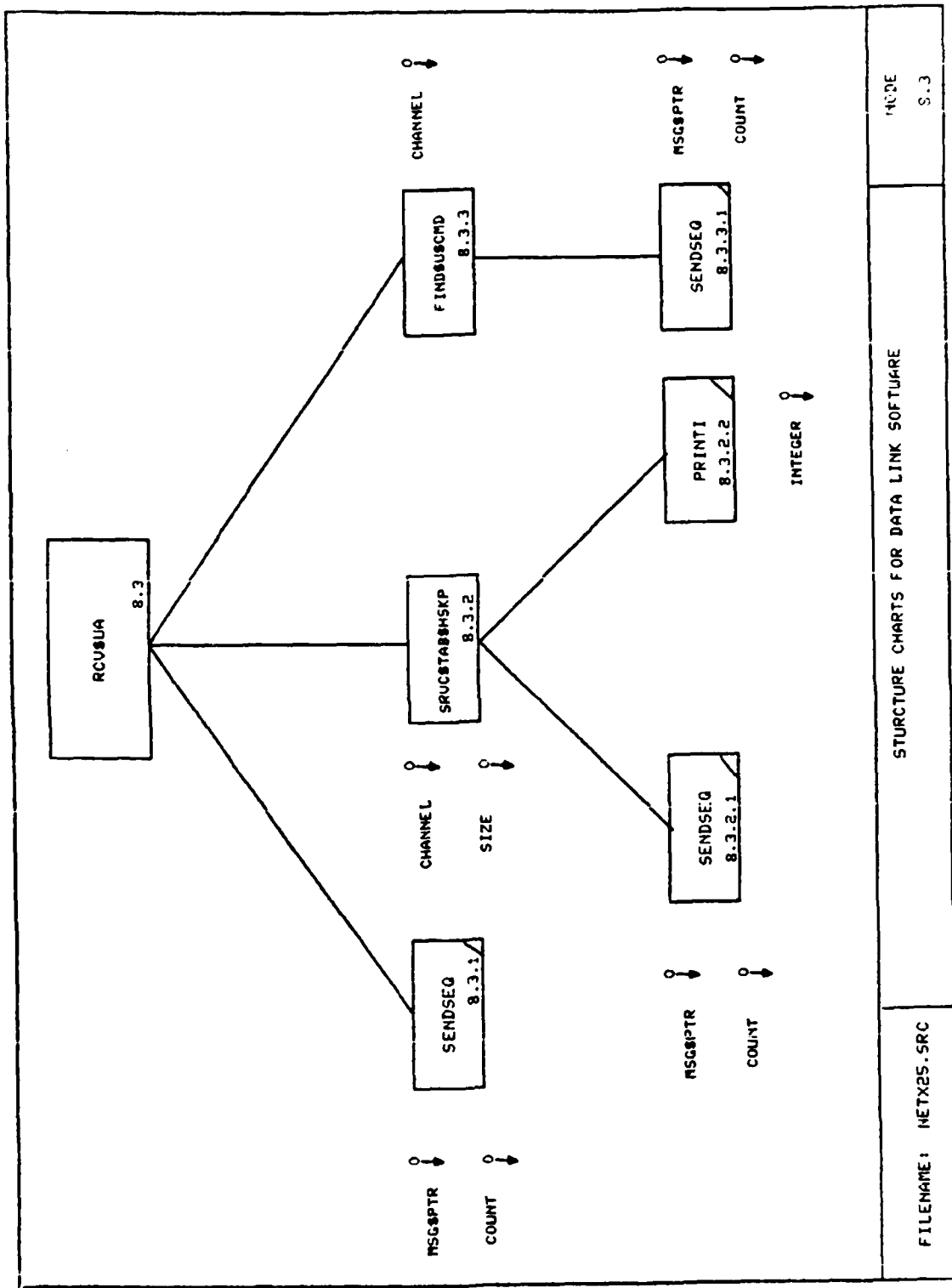
NODE  
8.0



FILENAME: NETX25.SRC	STRUCTURE CHARTS FOR DATA LINK SOFTWARE	INODE 8.2.2
----------------------	---	----------------



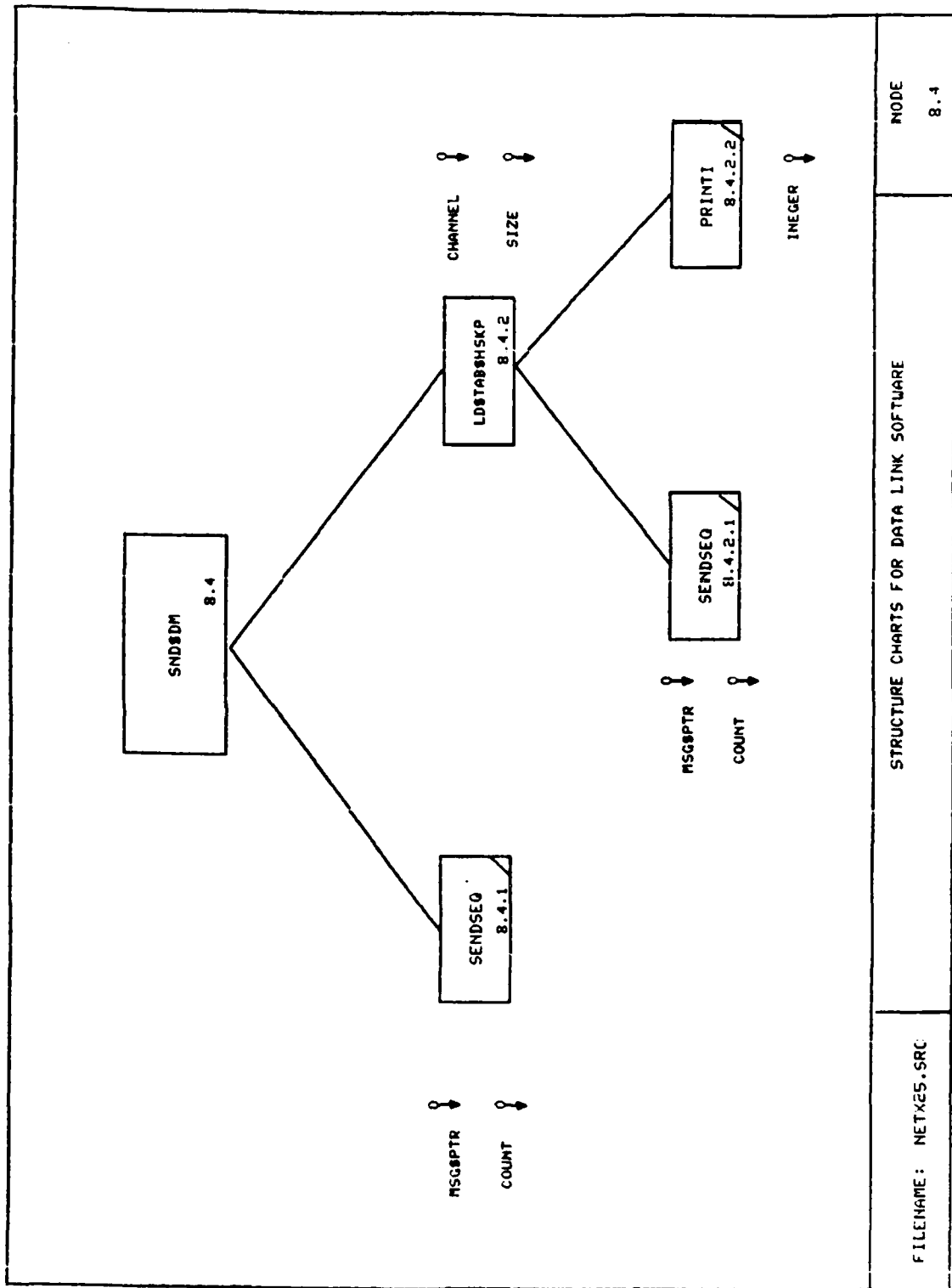
FILENAME: NETX25.SRC	STRUCTURE CHARTS FOR DATA LINK SOFTWARE	NODE 8.2.2
----------------------	---	---------------



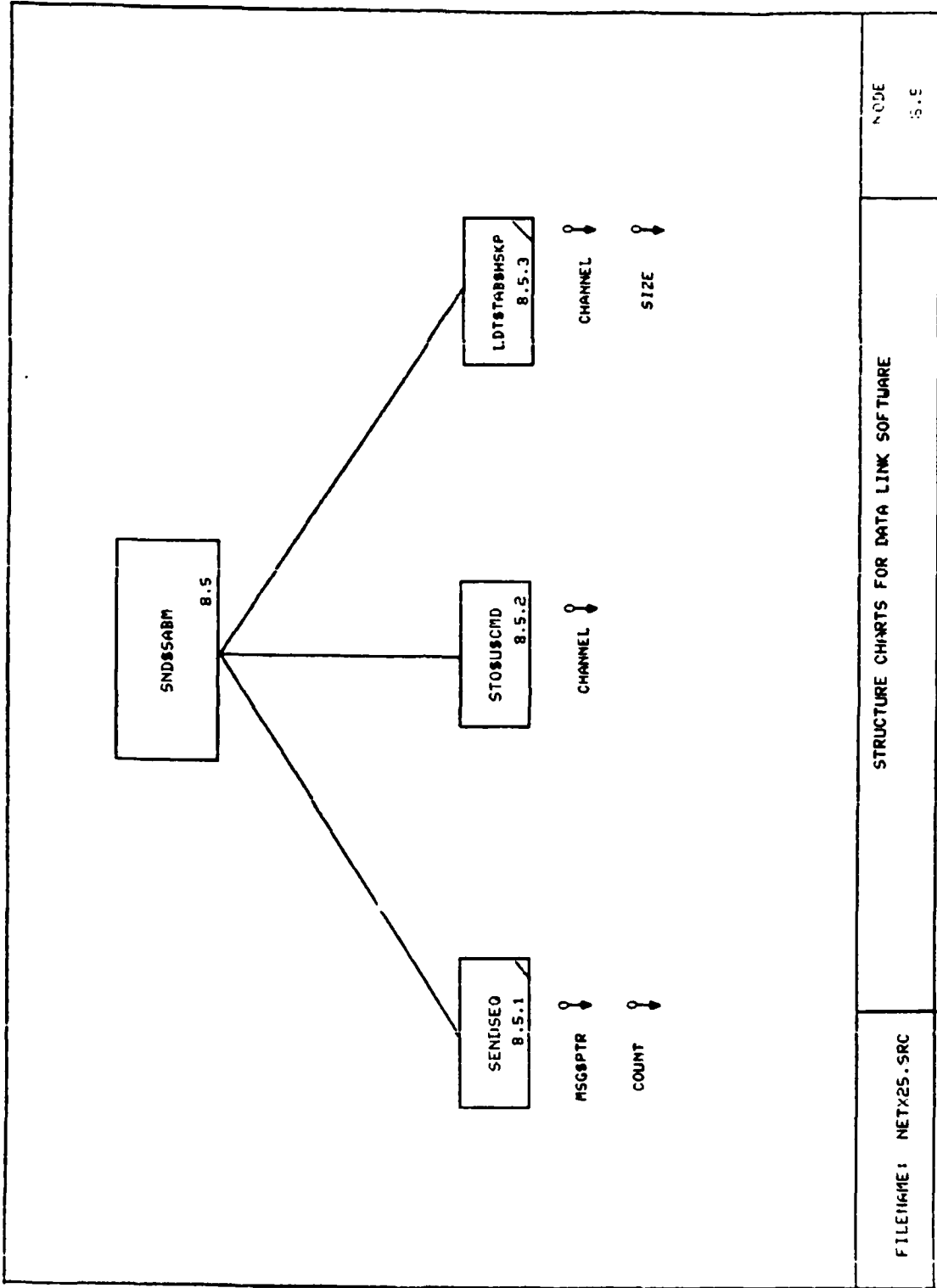
STRUCTURE CHARTS FOR DATA LINK SOFTWARE

FILENAME: NETX25.SRC

MODE  
S.3

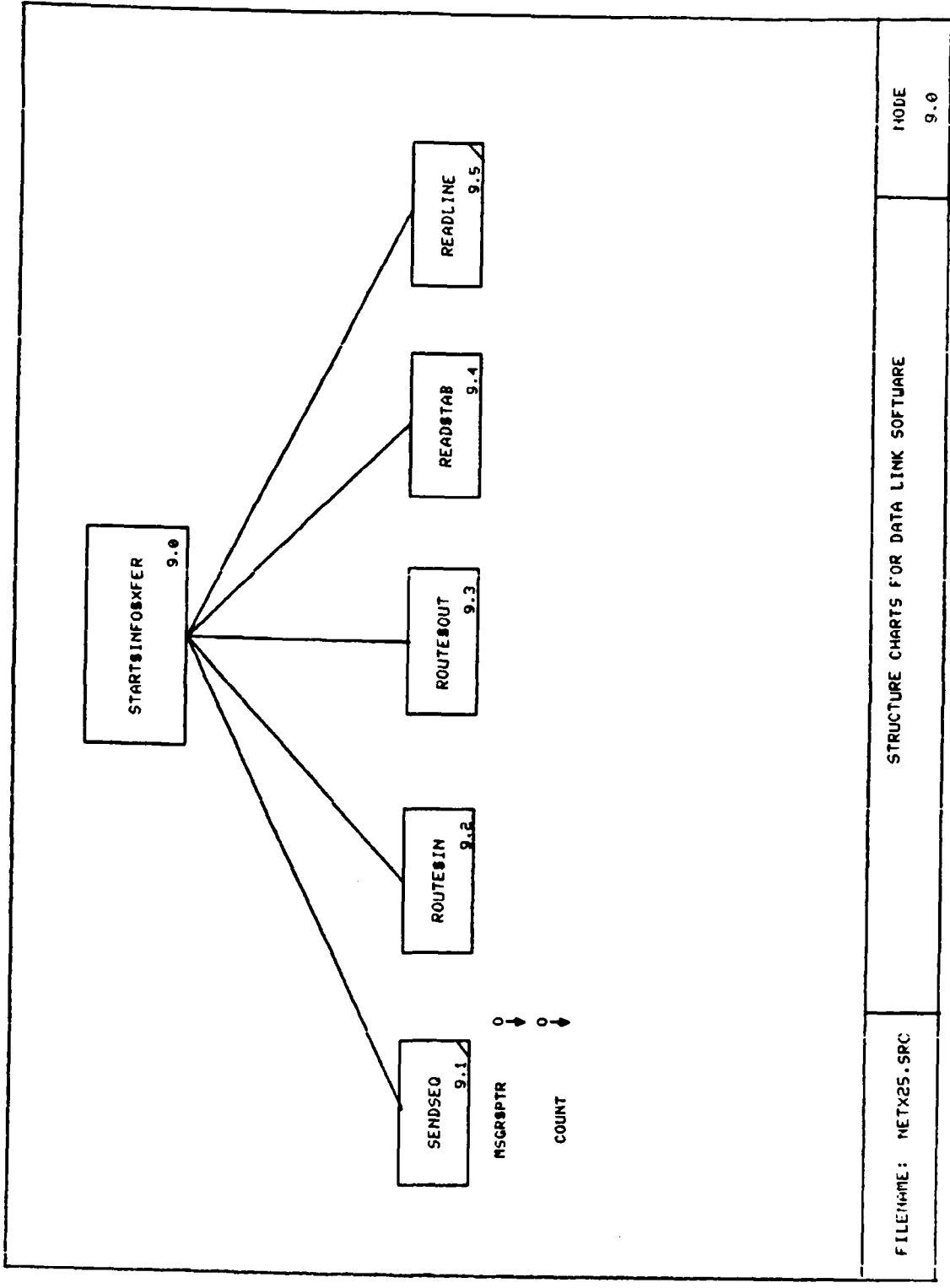


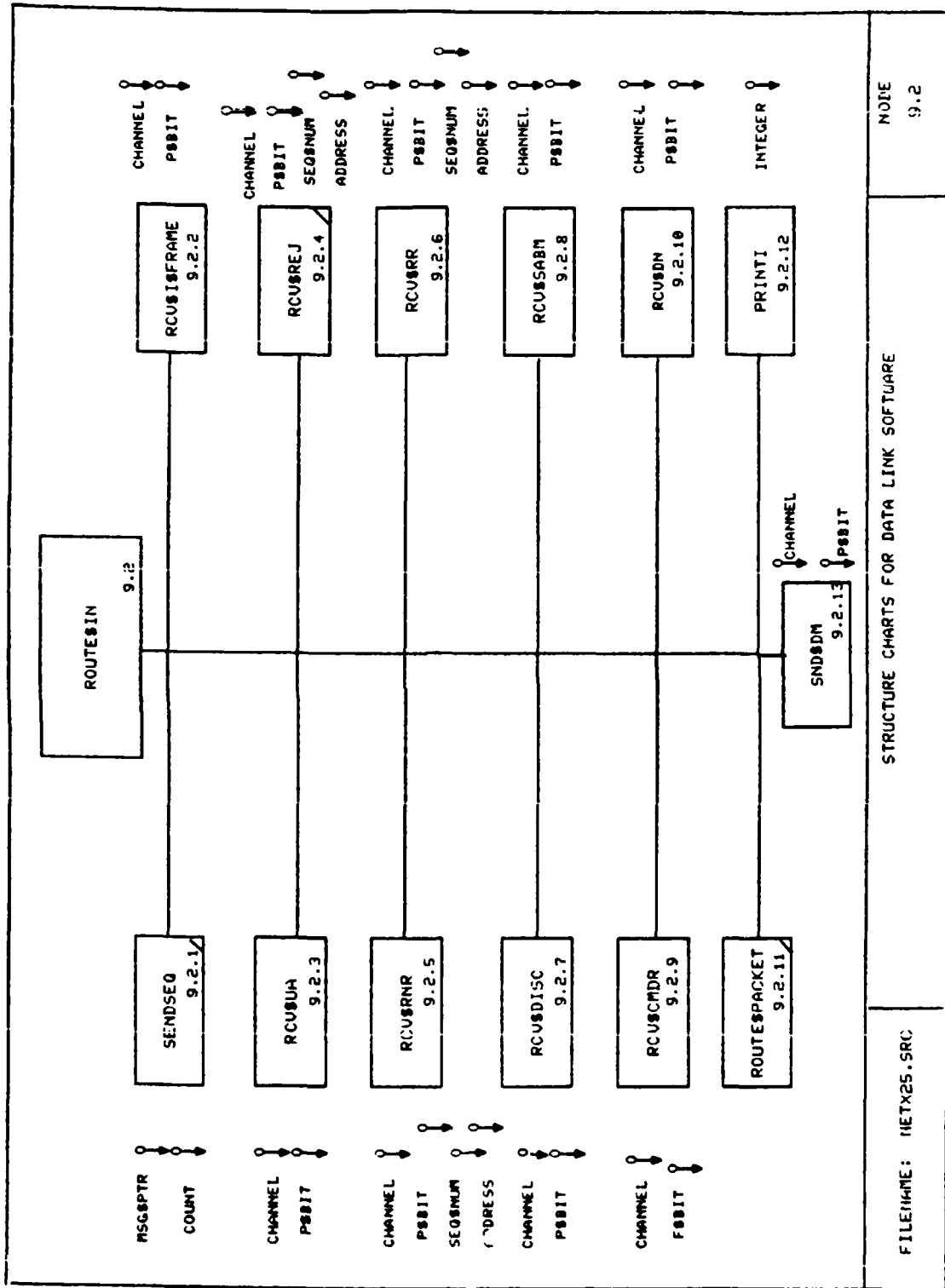
FILENAME: NETX25.SRC	STRUCTURE CHARTS FOR DATA LINK SOFTWARE	MODE 8.4
----------------------	---	-------------

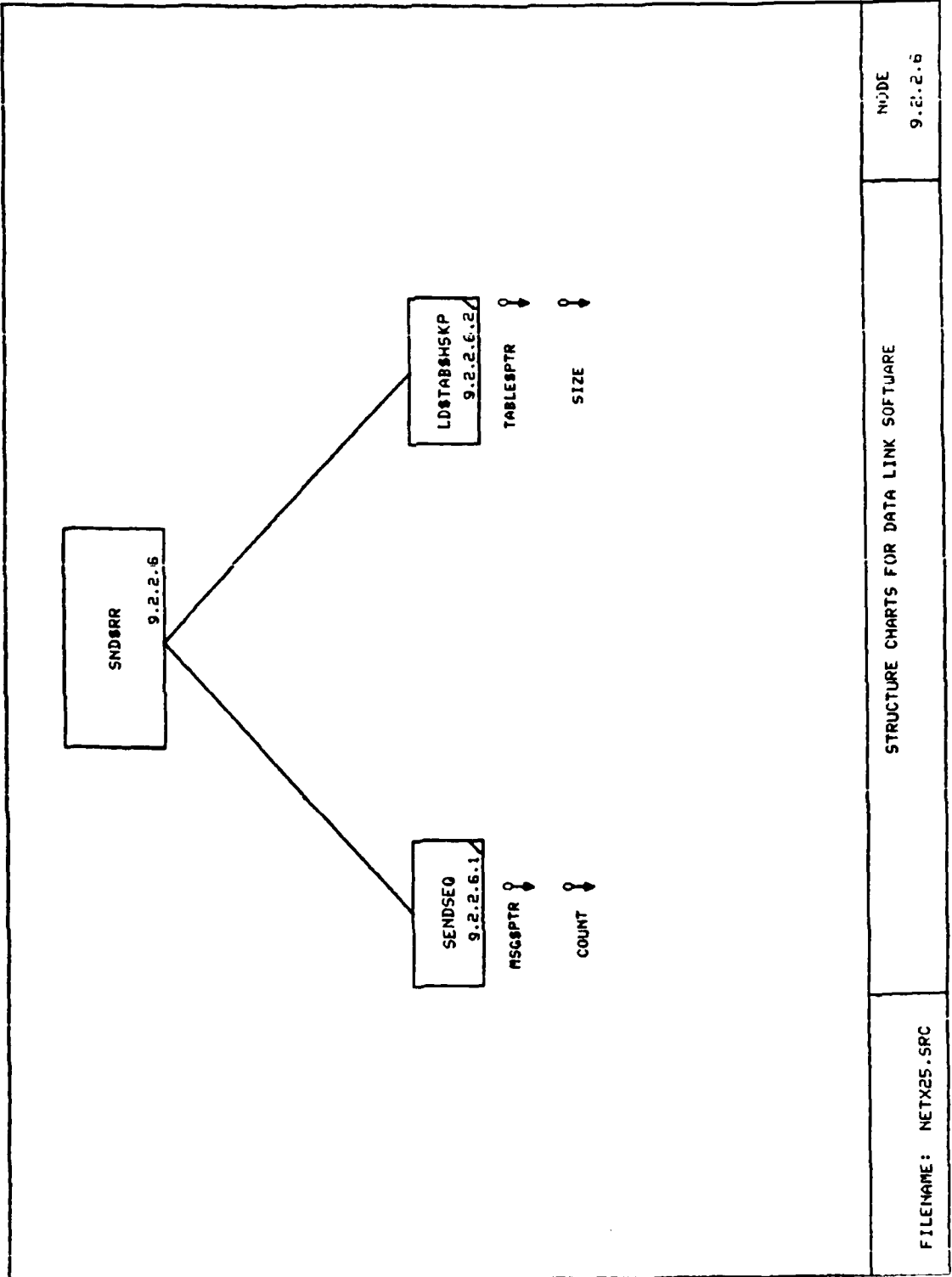


FILENAME: NETX25.SRC	STRUCTURE CHARTS FOR DATA LINK SOFTWARE	NODE 8.5
----------------------	---	-------------

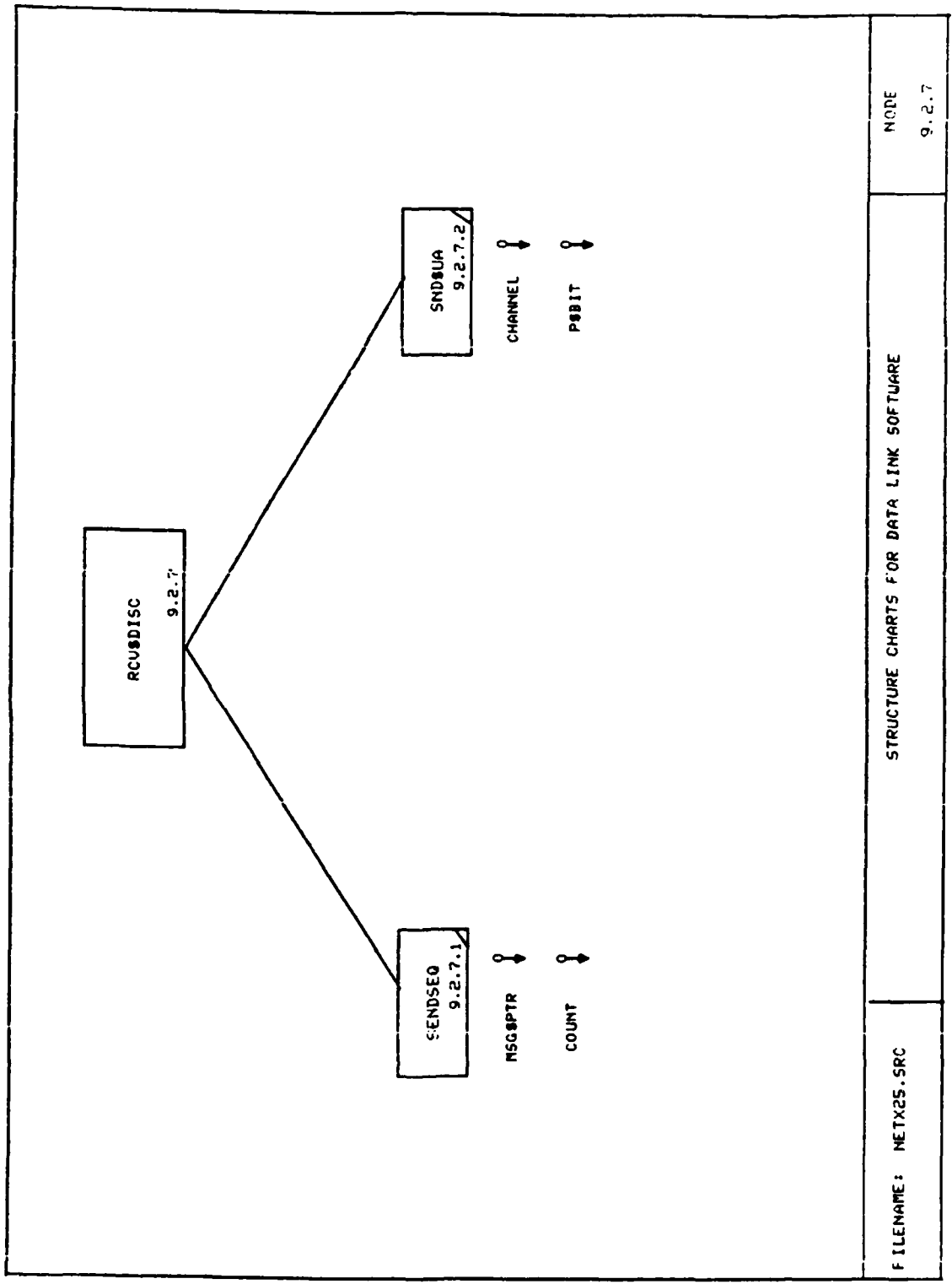








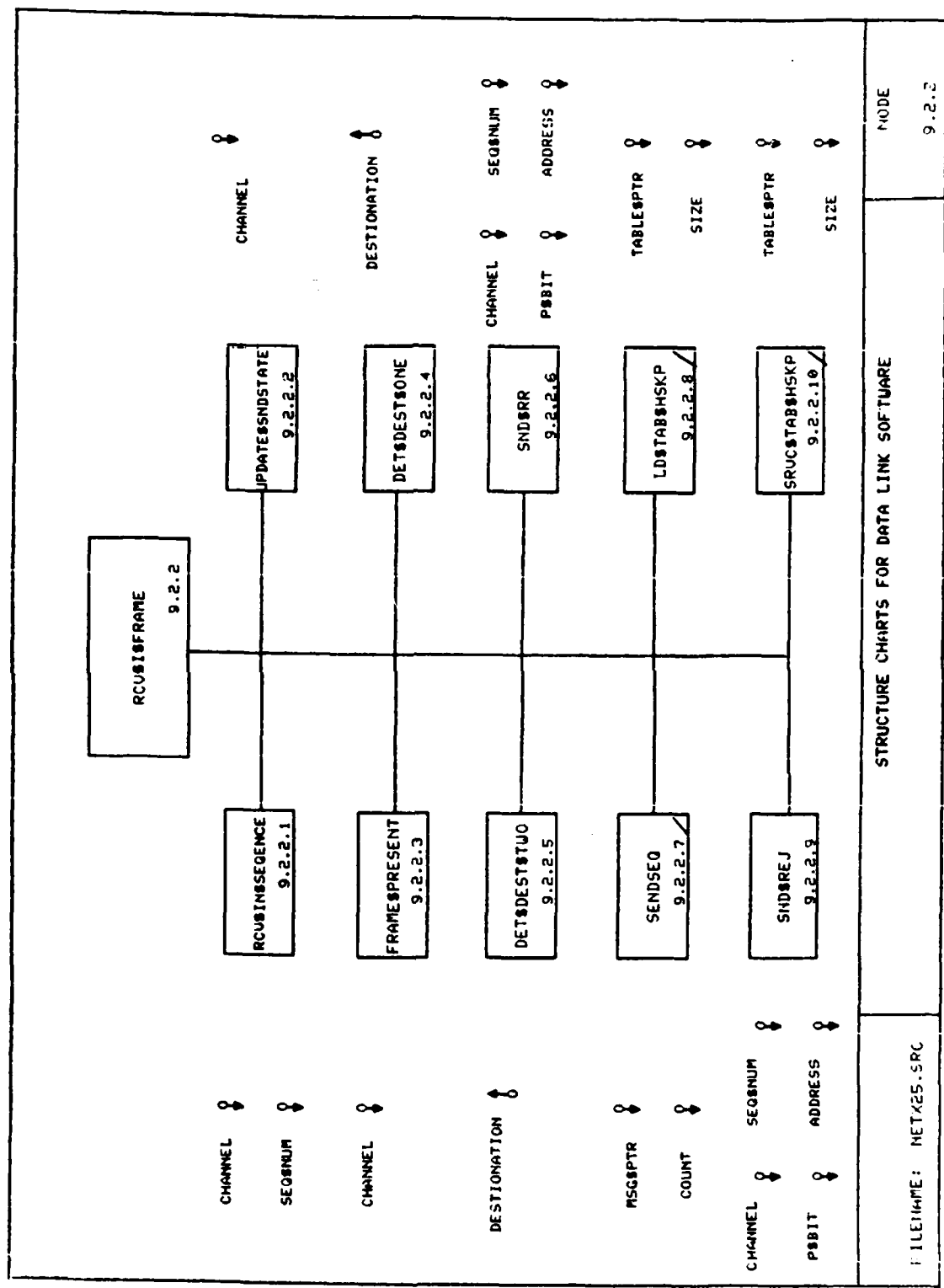
FILENAME: NETX25.SRC	STRUCTURE CHARTS FOR DATA LINK SOFTWARE	NODE 9.2.2.6
----------------------	---	-----------------



FILENAME: NETX25.SRC

STRUCTURE CHARTS FOR DATA LINK SOFTWARE

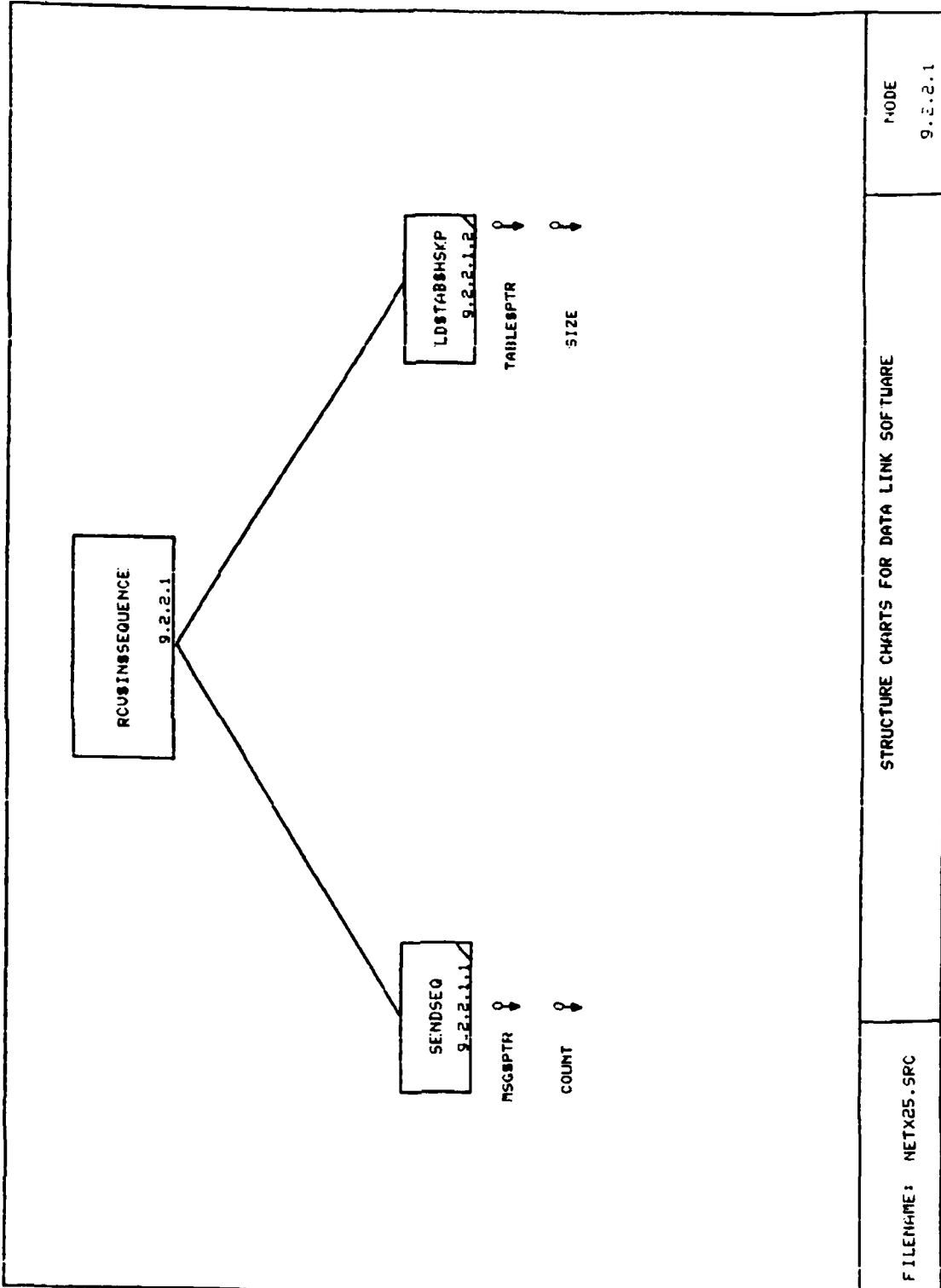
NODE  
9.2.7



STRUCTURE CHARTS FOR DATA LINK SOFTWARE

FILENAME: NETX25.SRC

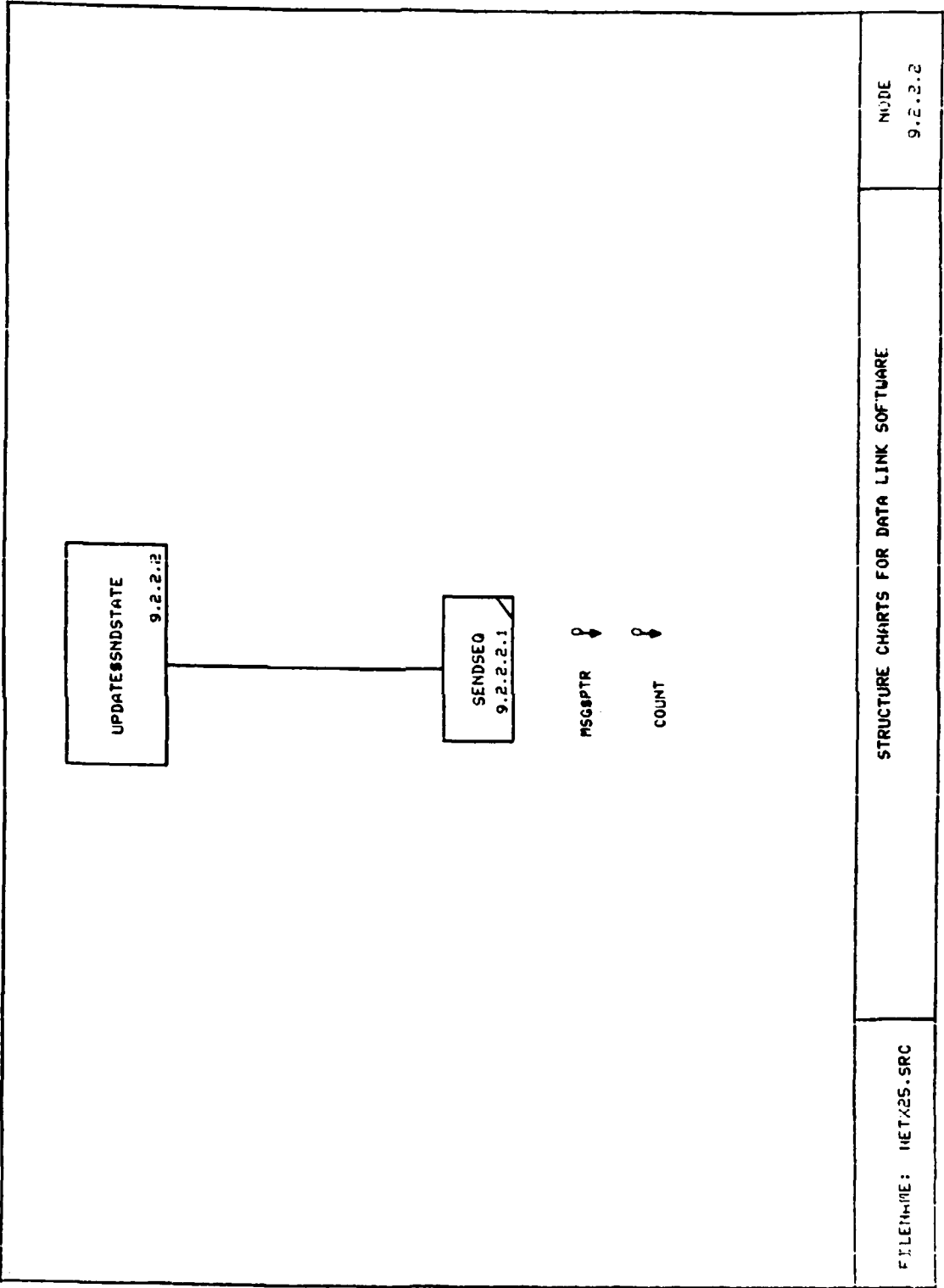
NODE  
9.2.2



MODE  
9.2.2.1

STRUCTURE CHARTS FOR DATA LINK SOFTWARE

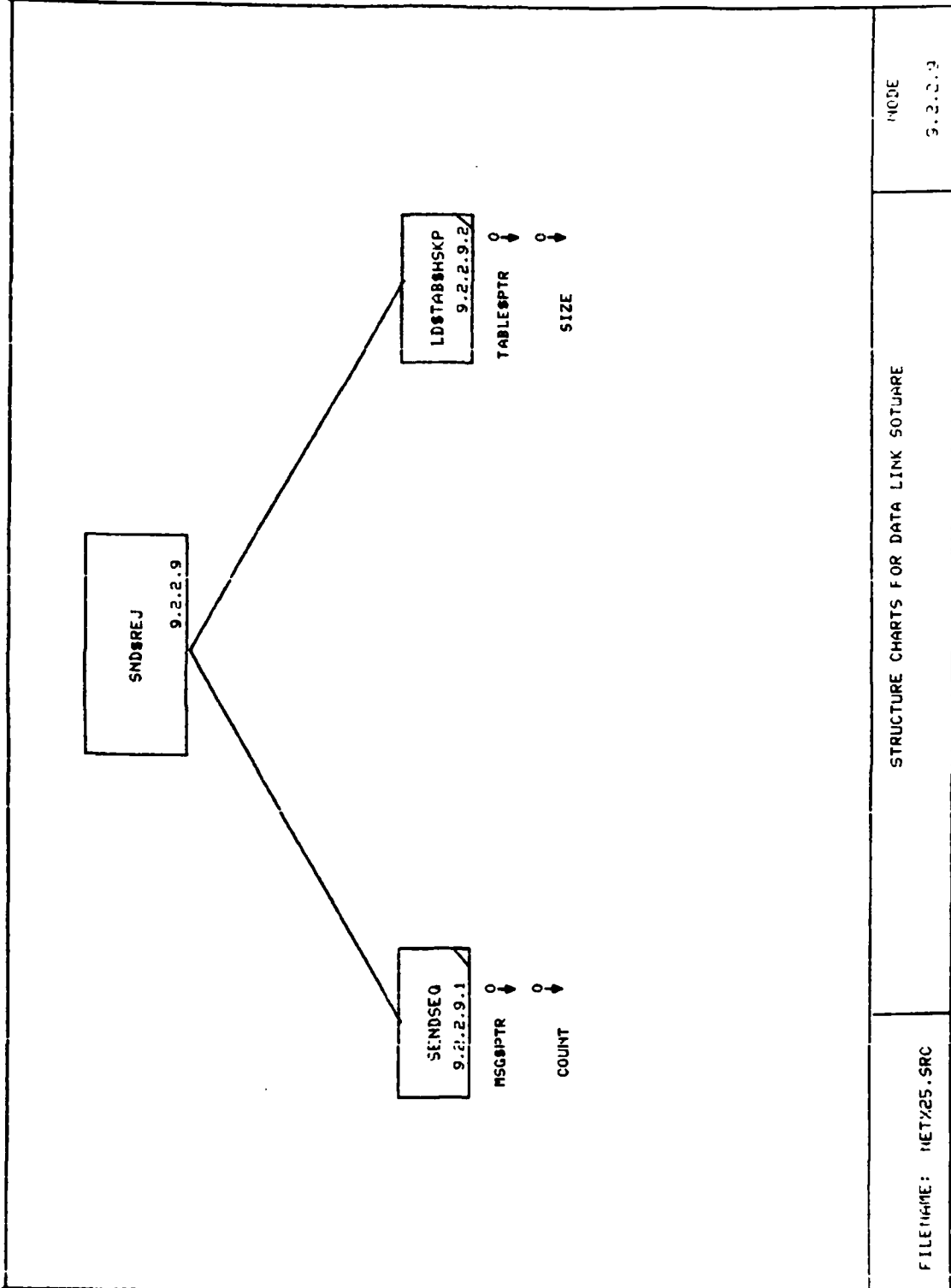
FILENAME: NETX25.SRC



NODE  
9.2.2.2

STRUCTURE CHARTS FOR DATA LINK SOFTWARE

FILENAME: NETAB5.SRC

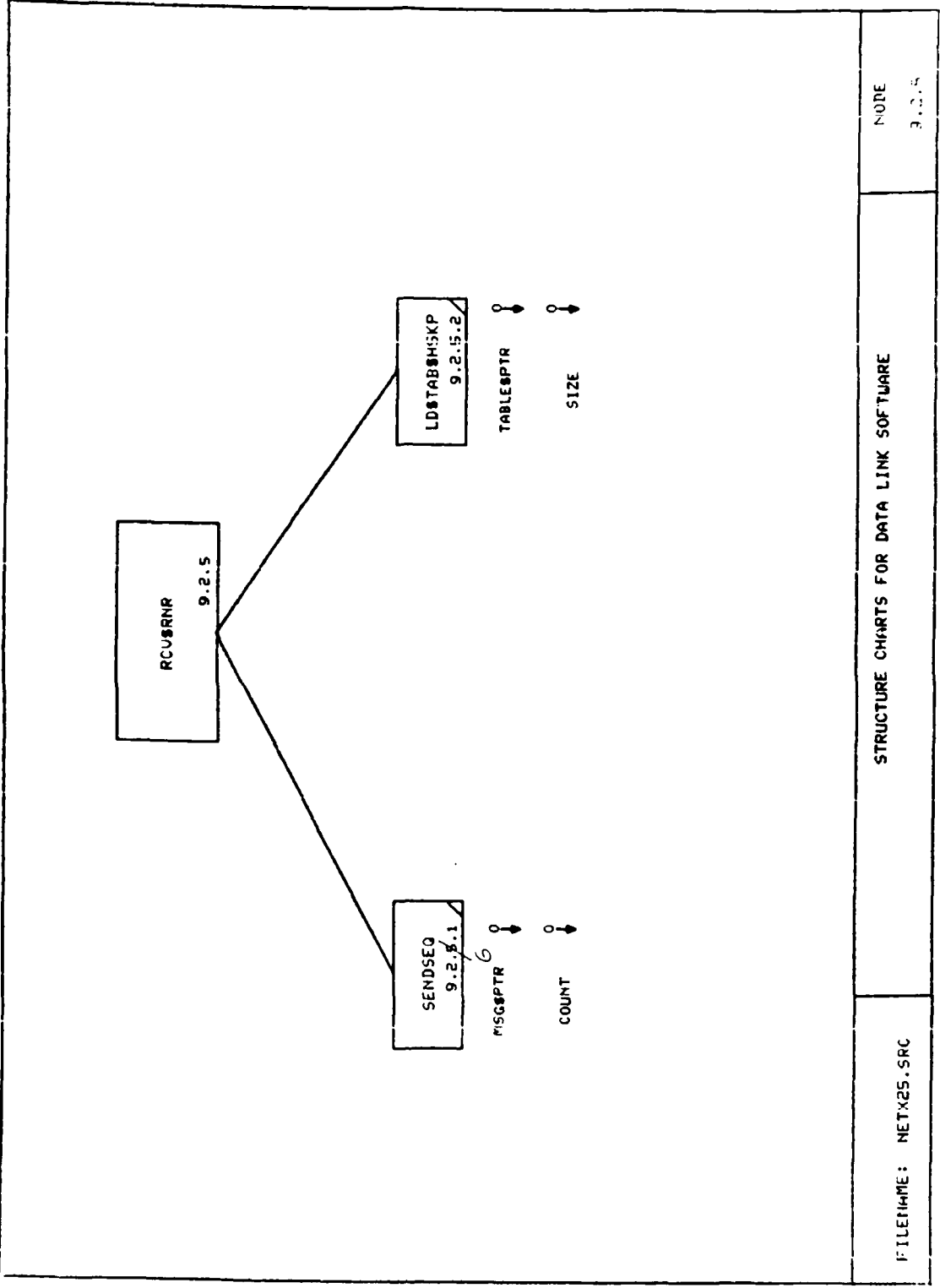


STRUCTURE CHARTS FOR DATA LINK SOFTWARE

FILENAME: NETX25.SRC

MODE  
9.2.2.9



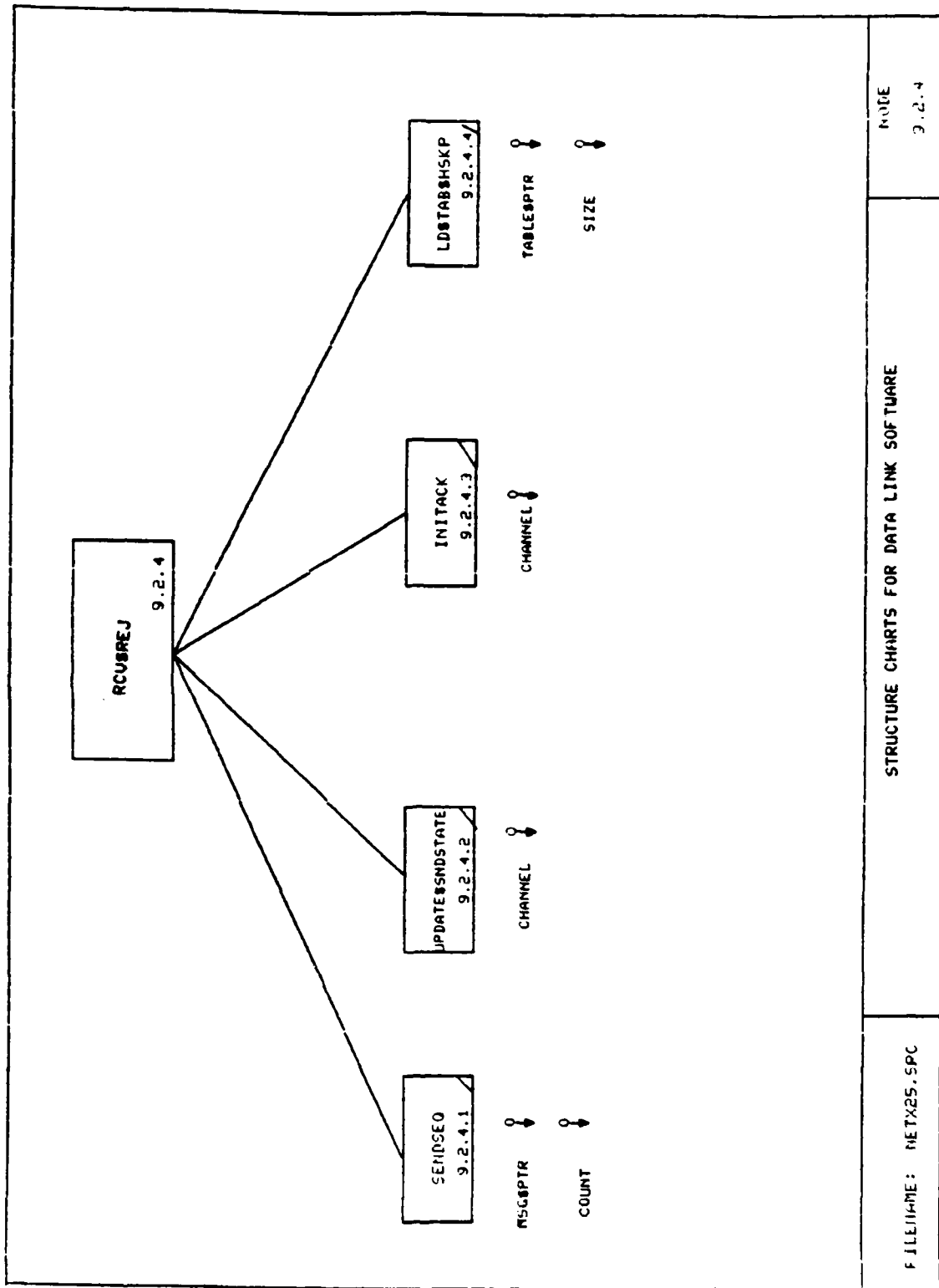


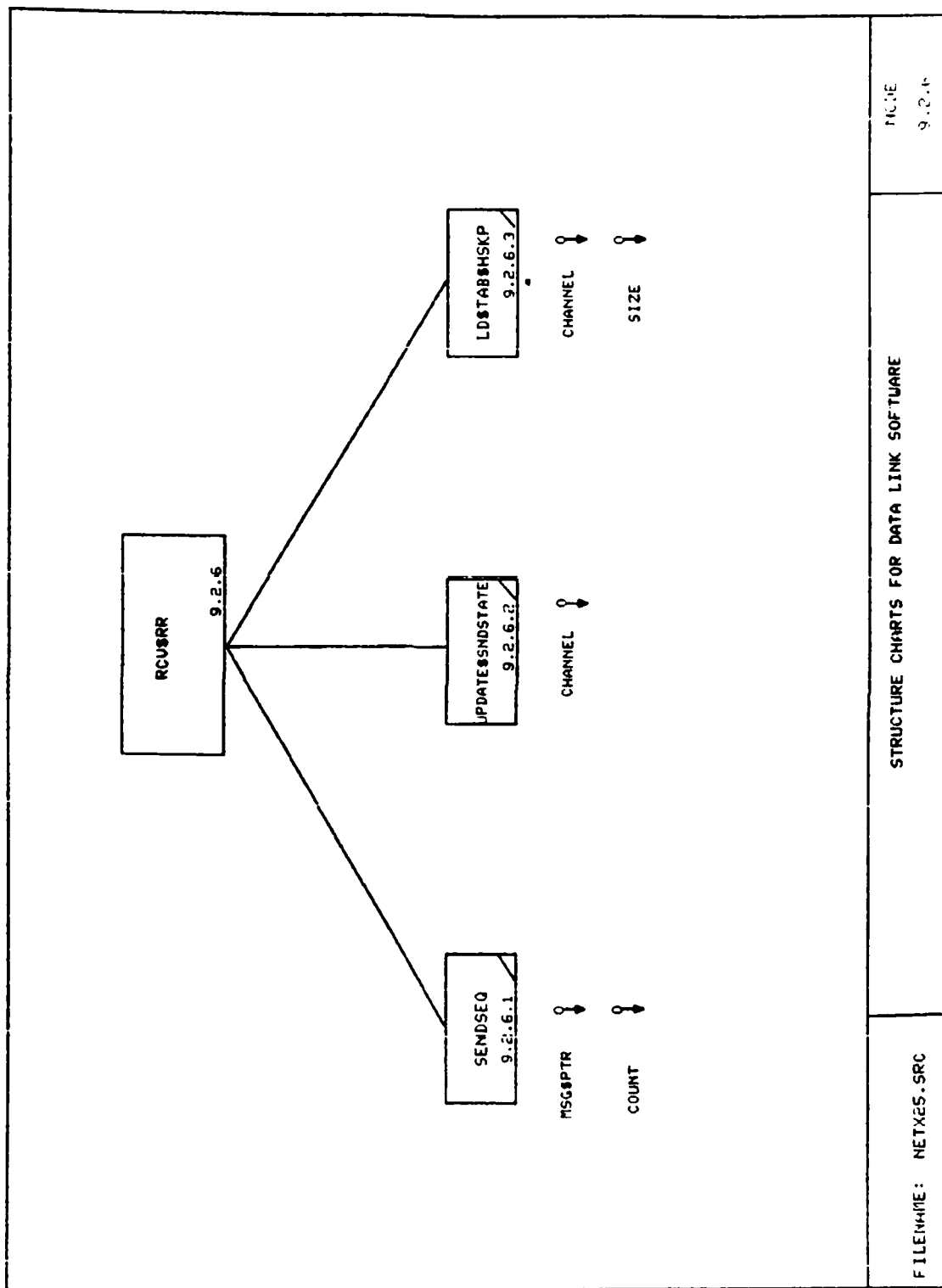
NOTE

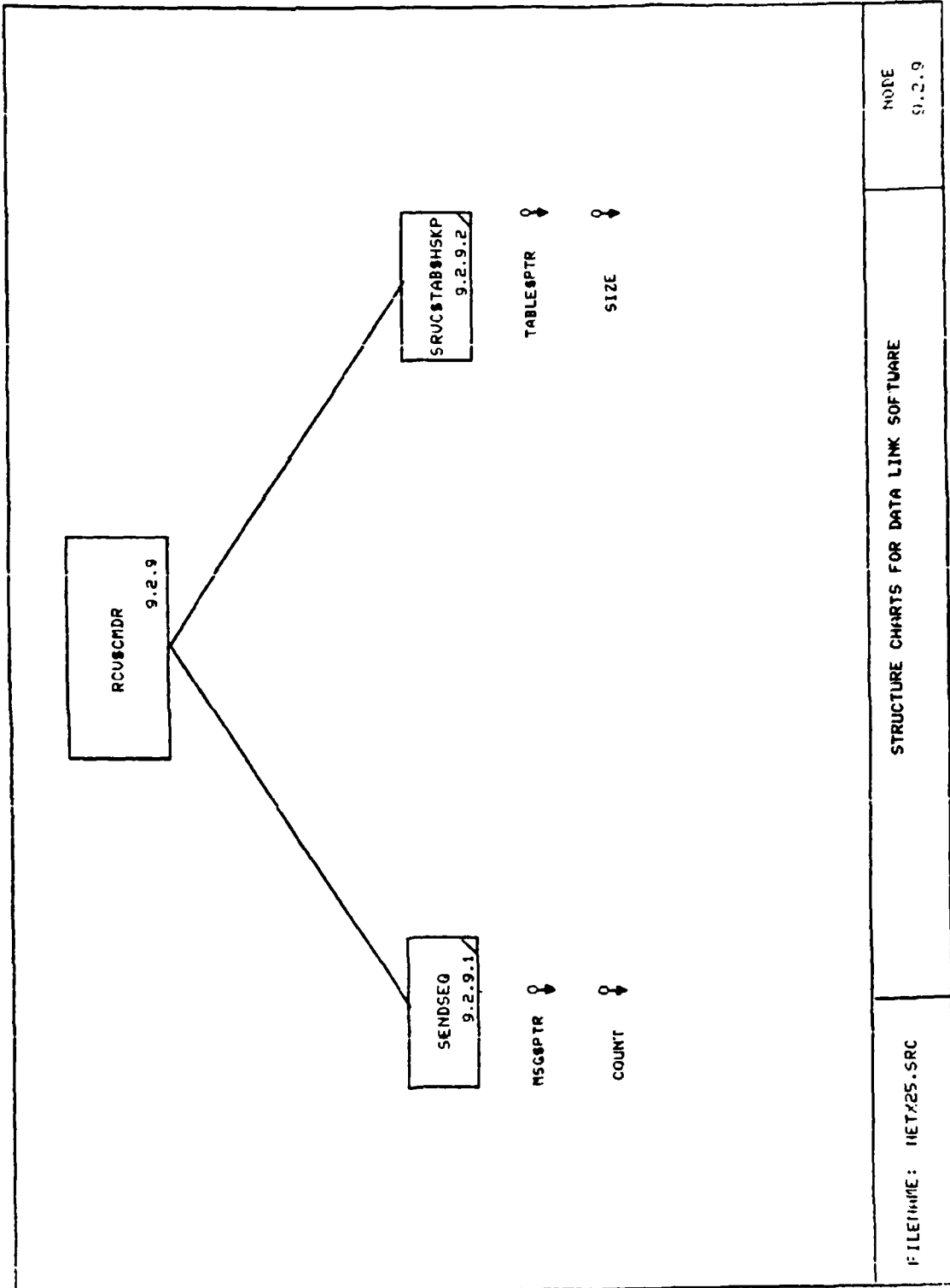
3.2.5

STRUCTURE CHARTS FOR DATA LINK SOFTWARE

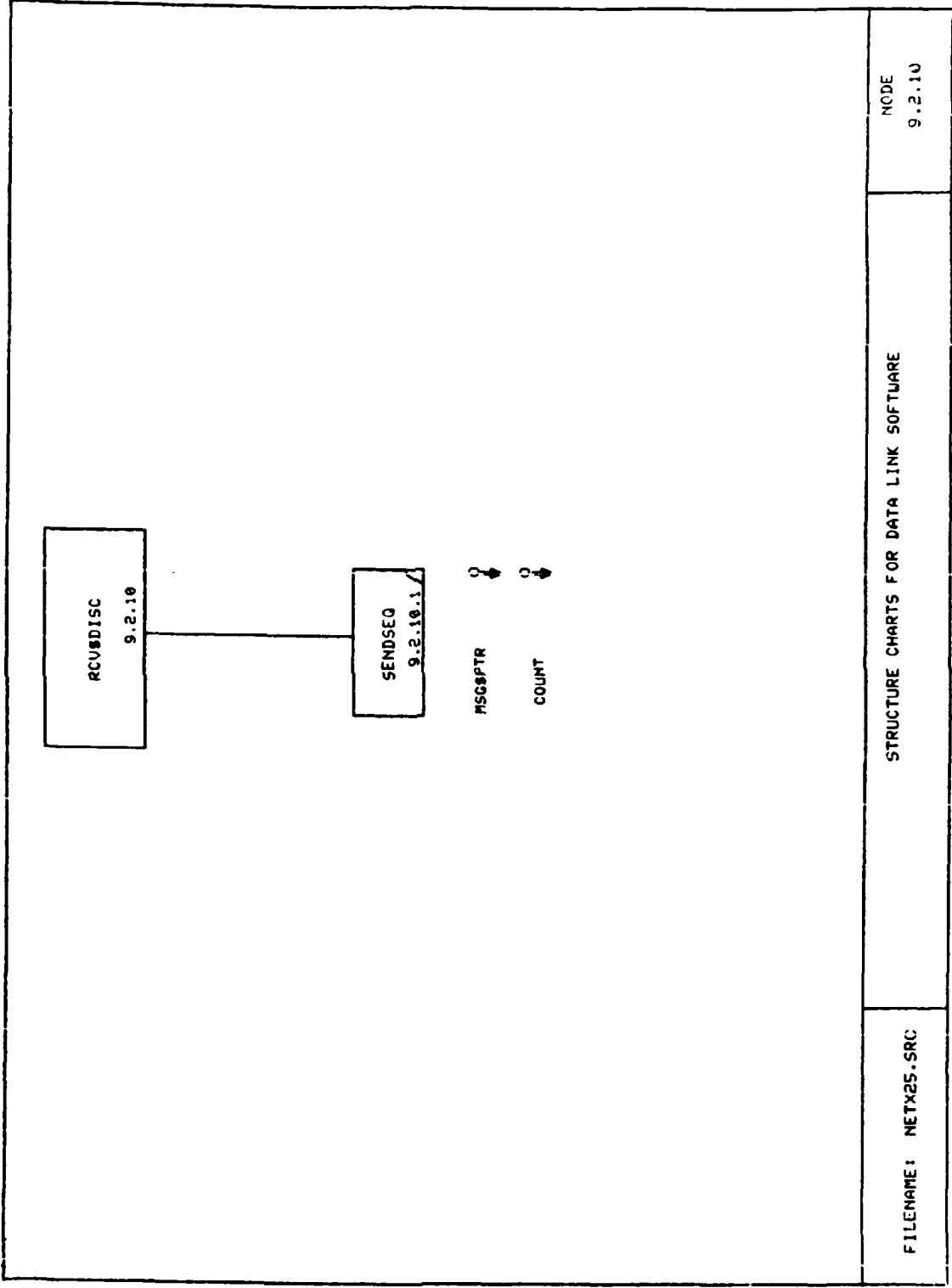
FILENAME: NETX25.SRC







FILENAME: NETX25.SRC	STRUCTURE CHARTS FOR DATA LINK SOFTWARE	NODE 9.2.9
----------------------	---	---------------

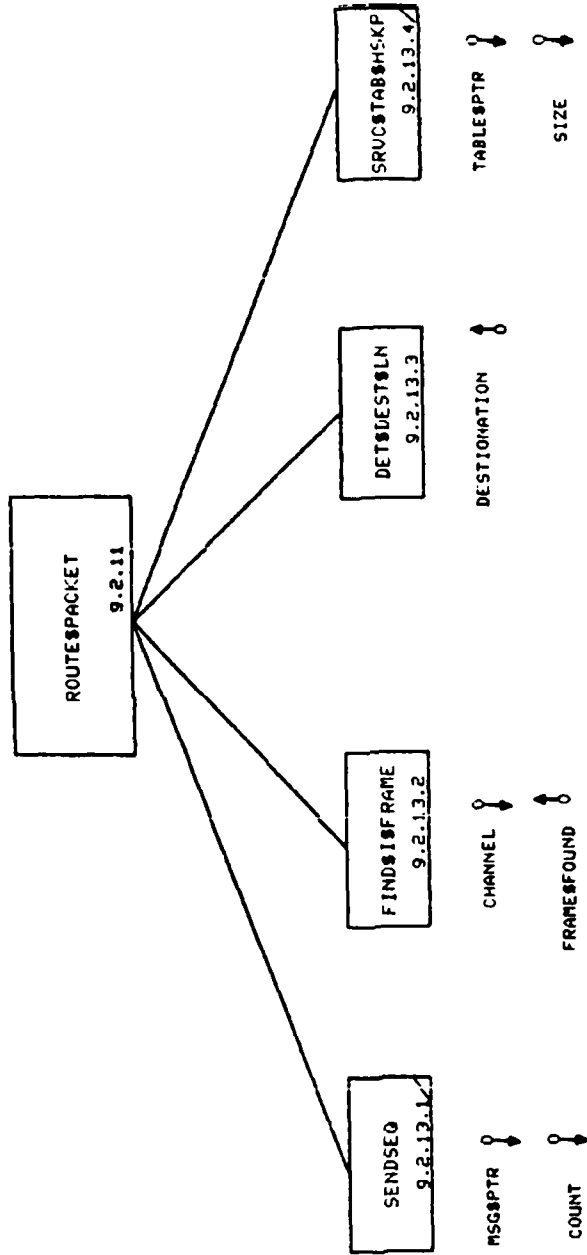


NODE  
9.2.10

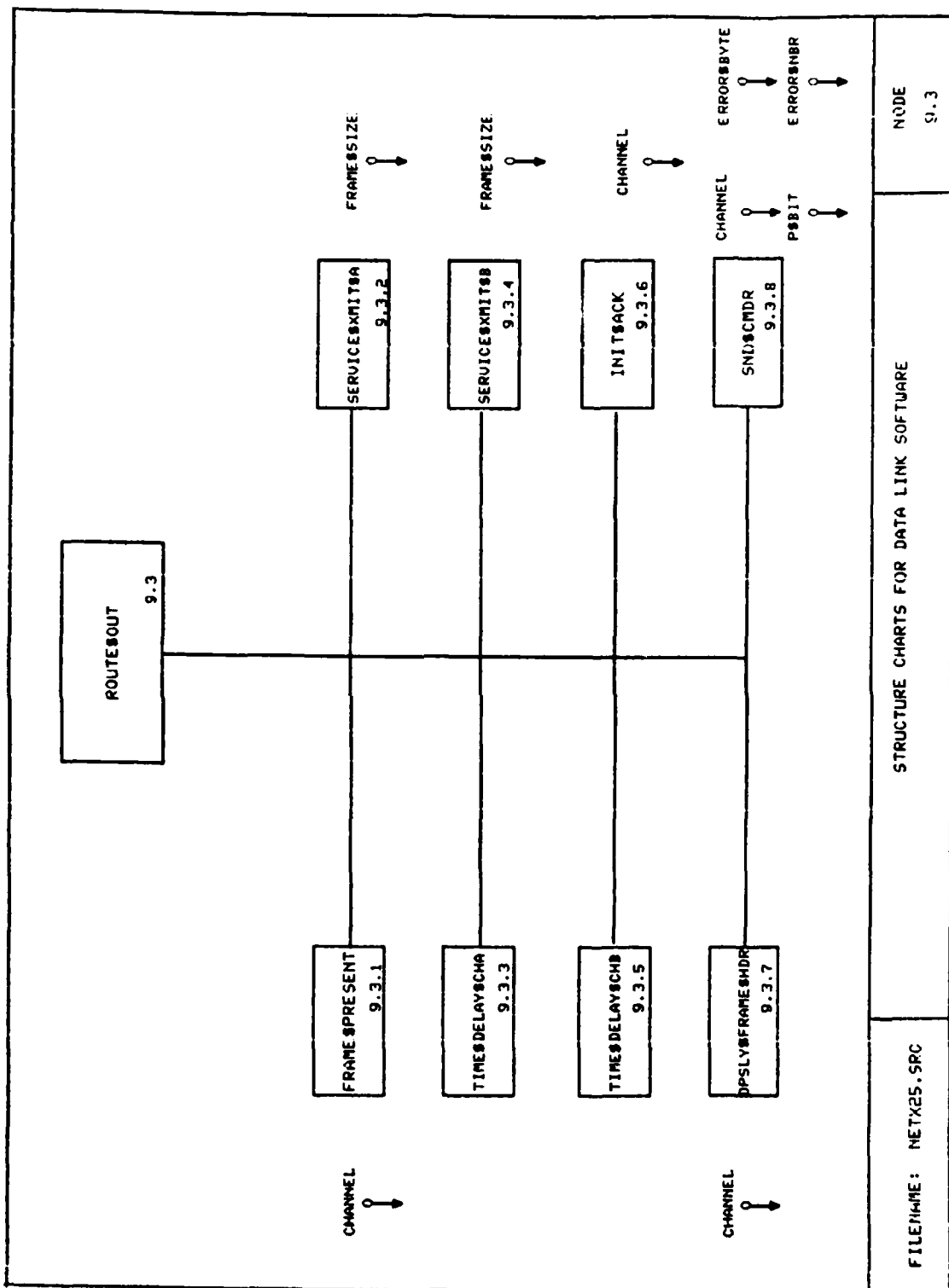
STRUCTURE CHARTS FOR DATA LINK SOFTWARE

FILENAME: NETX25.SRC

TYPE7 ->U



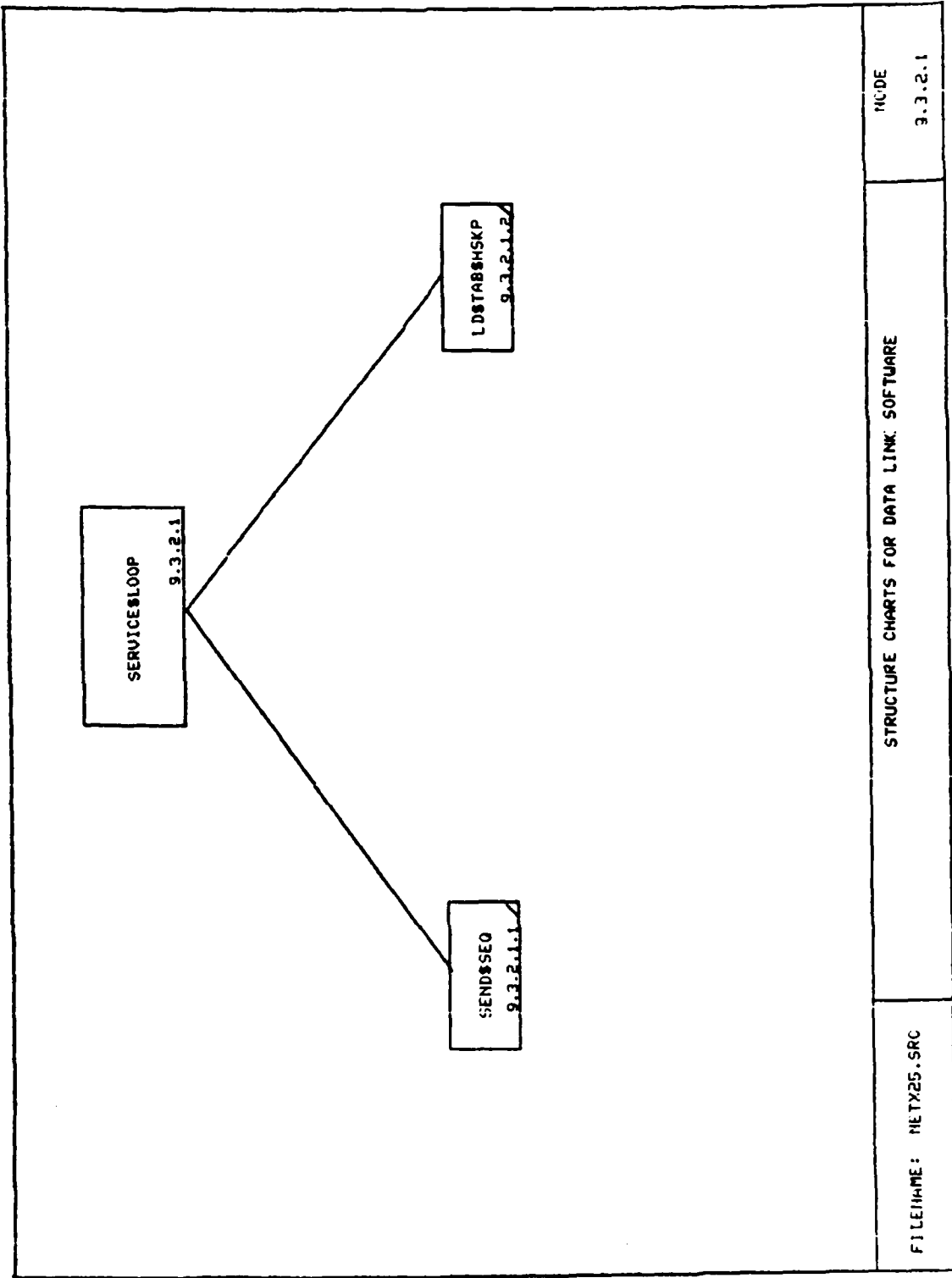
FILENAME: NIETY25.SRC	STRUCTURE CHARTS FOR DATA LINK SOFTWARE	NODE 9.2.11
-----------------------	---	----------------



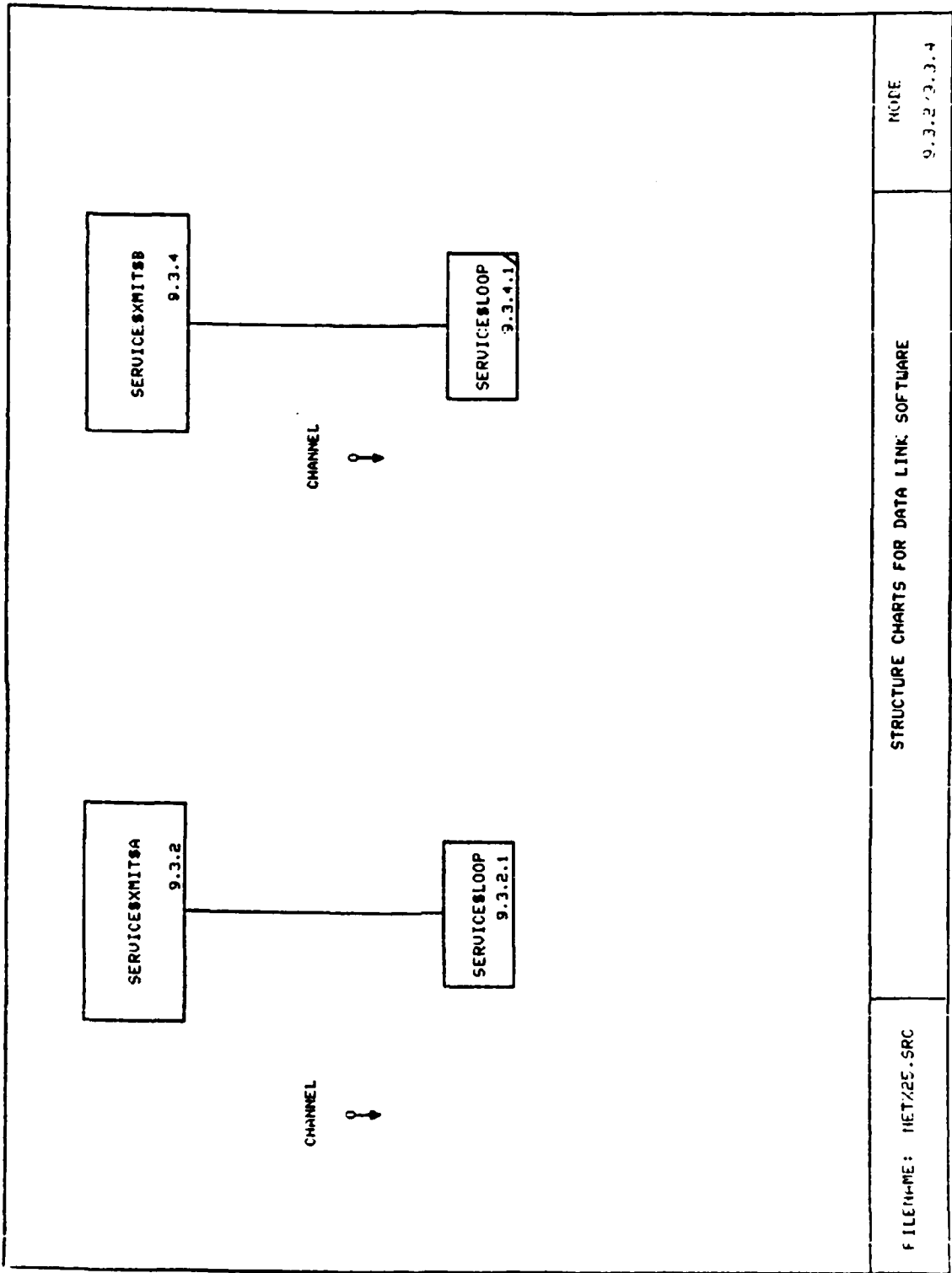
STRUCTURE CHARTS FOR DATA LINK SOFTWARE

FILENAME: NETX25.SRC

NODE  
9.3.3



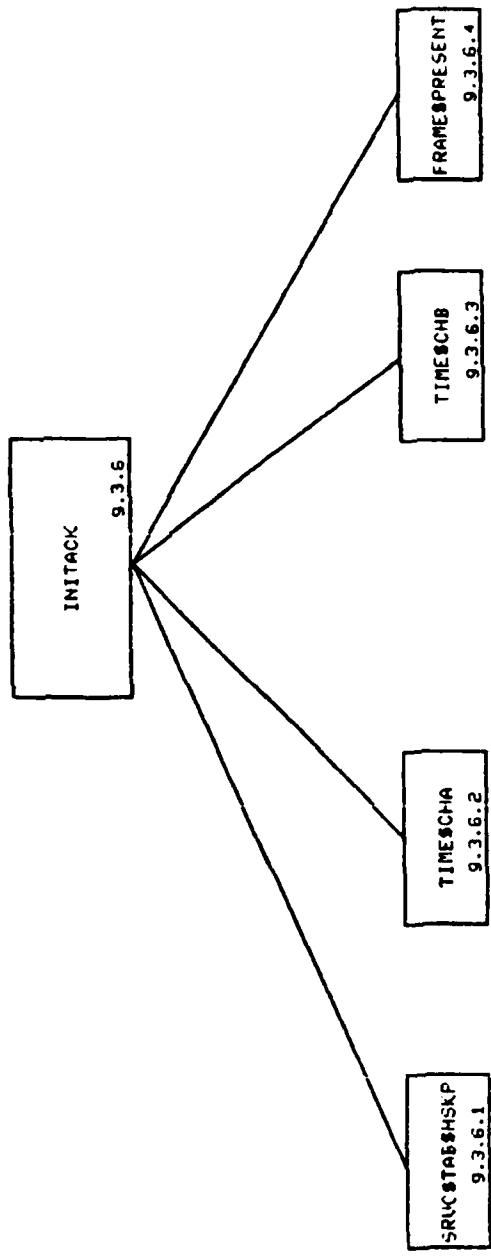




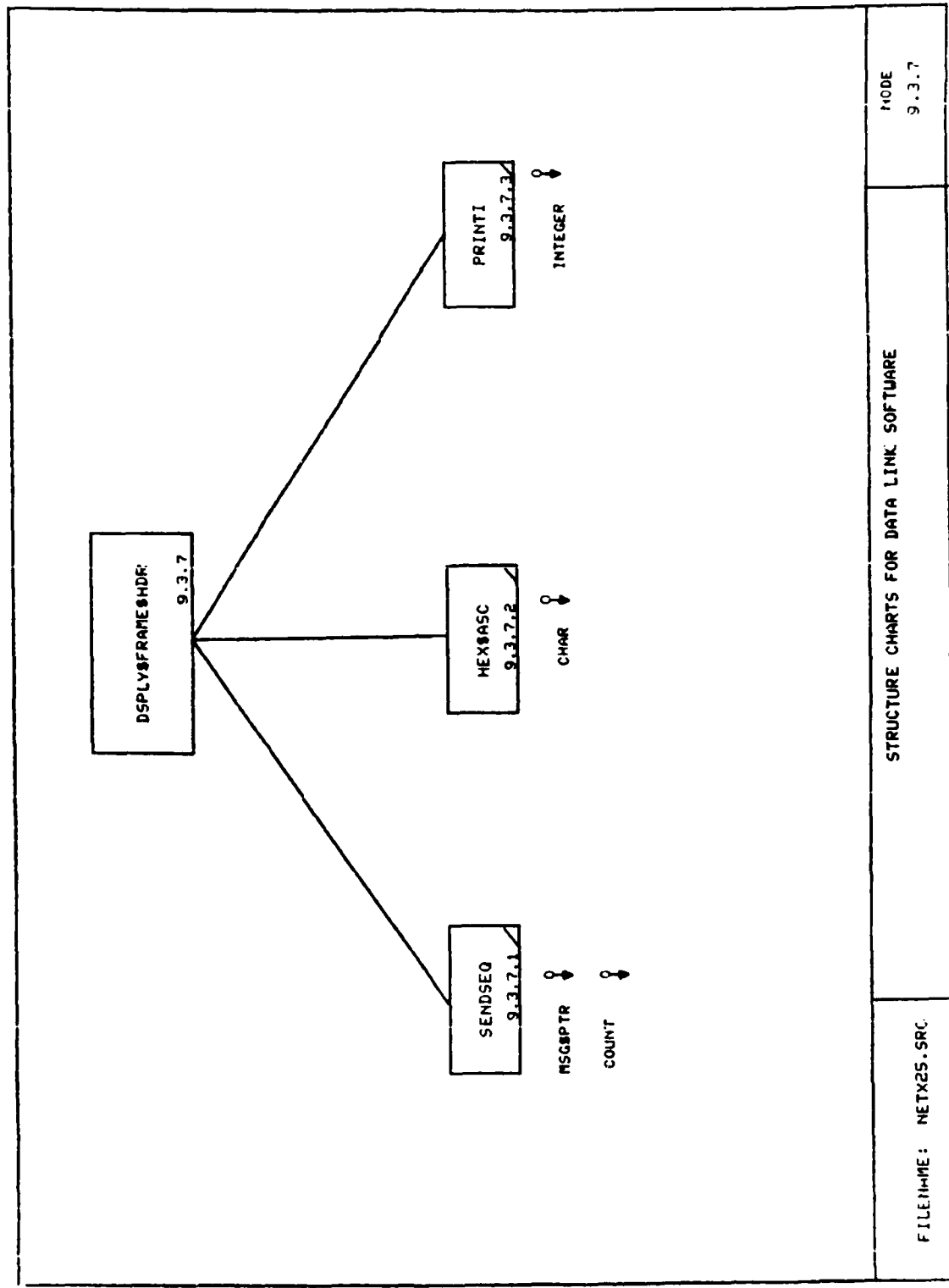
MODE  
9.3.2/3.3.4

STRUCTURE CHARTS FOR DATA LINK SOFTWARE

FILE NAME: NET/25.SRC



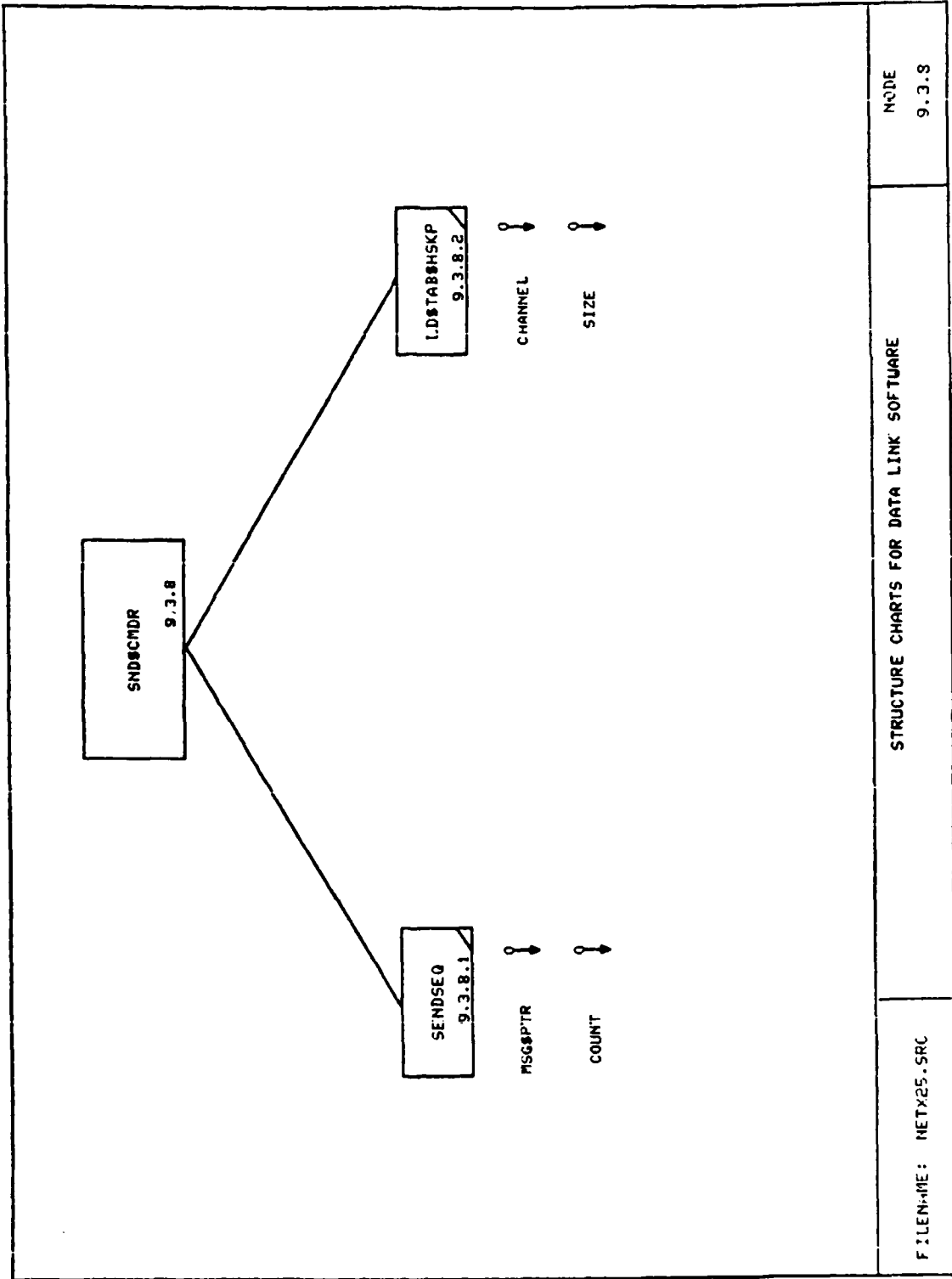
FILENAME: HET/25.SPC	STRUCTURE CHARTS FOR DATA LINK SOFTWARE	NODE 9.3.6
----------------------	---	---------------



MODE  
9.3.7

STRUCTURE CHARTS FOR DATA LINK SOFTWARE

FILENAME: NETX25.SRC



FILENAME: NETX25.SRC	STRUCTURE CHARTS FOR DATA LINK SOFTWARE	NODE 9.3.8
----------------------	---	---------------

AFIT/GE/ENG/85D-52

DEVELOPMENT AND IMPLEMENTATION  
OF THE  
X.25 PROTOCOL  
FOR THE  
UNIVERSAL NETWORK INTERFACE DEVICE (UNID) II  
VOL II OF II

THESIS

AFIT/GE/ENG/85D-52

Mark W. Weber  
Captain

USAF

DEVELOPMENT AND IMPLEMENTATION  
OF THE  
X.25 PROTOCOL  
FOR THE  
UNIVERSAL NETWORK INTERFACE DEVICE (UNID) II  
VOL II OF II

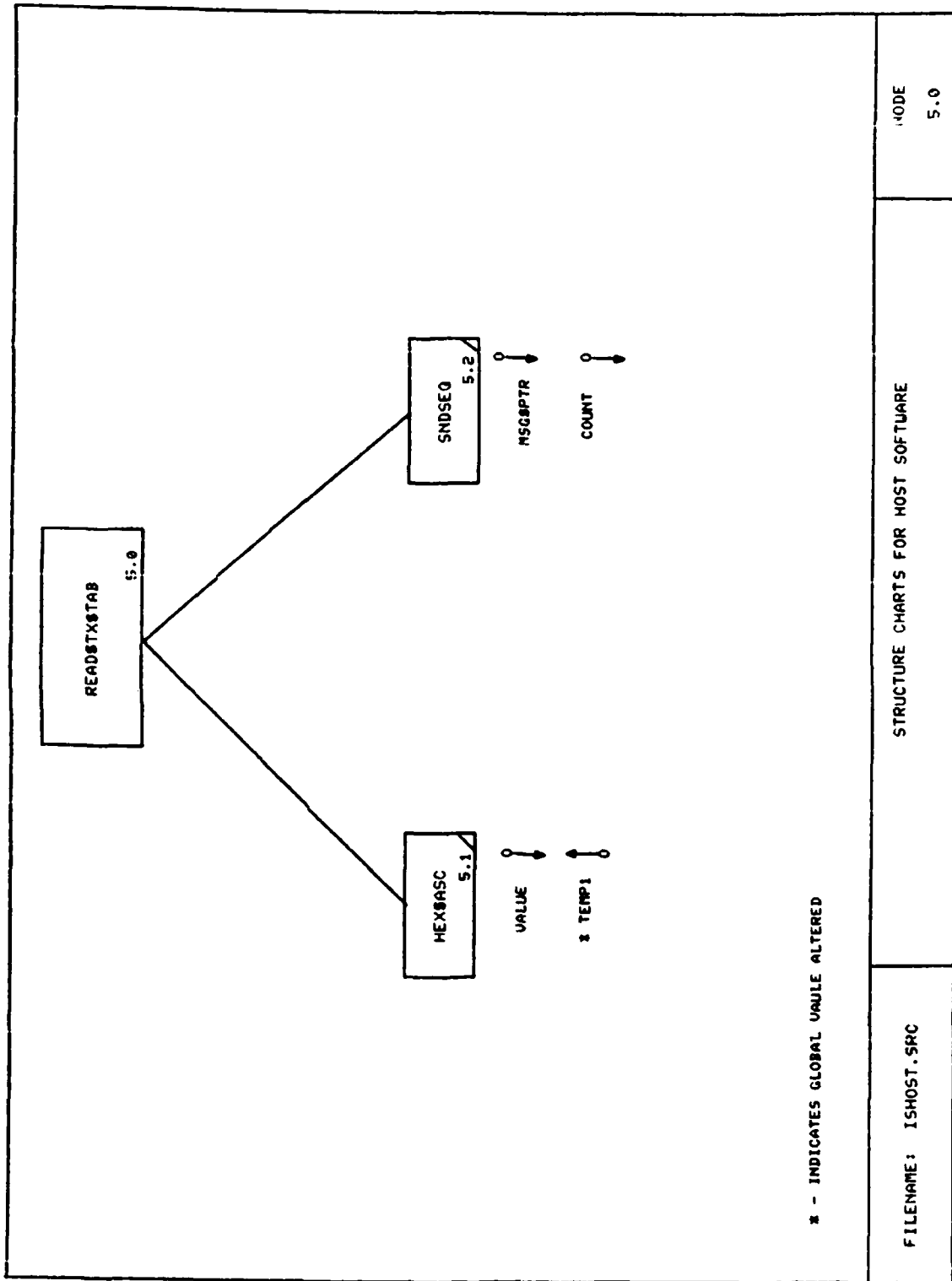
THESIS

Presented to the Faculty of the School of Engineering  
of the Air Force Institute of Technology  
Air University  
in Partial Fulfillment of the  
Requirements for the Degree of  
Master of Science in Electrical Engineering

Mark W. Weber, BS EE

Captain, USAF

December 1985

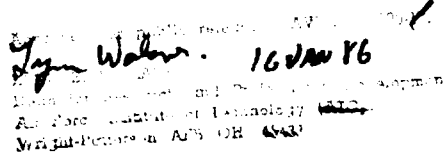


UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

AD-A16-4 C76

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION <b>UNCLASSIFIED</b>		1b. RESTRICTIVE MARKINGS	
2a. SECURITY CLASSIFICATION AUTHORITY		3. DISTRIBUTION/AVAILABILITY OF REPORT	
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE		Approved for public release; <u>distribution unlimited.</u>	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)  AFIT/GE/85D-52		5. MONITORING ORGANIZATION REPORT NUMBER(S)	
6a. NAME OF PERFORMING ORGANIZATION School of Engineering	6b. OFFICE SYMBOL (If applicable) AFIT/EN	7a. NAME OF MONITORING ORGANIZATION	
6c. ADDRESS (City, State and ZIP Code) Air Force Institute of Technology Wright Patterson AFB, OH 45433		7b. ADDRESS (City, State and ZIP Code)	
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Rome Air Development Ctr	8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER	
8c. ADDRESS (City, State and ZIP Code) Griffiss AFB NY 13441		10. SOURCE OF FUNDING NOS.	
11. TITLE (Include Security Classification) See Box 19		PROGRAM ELEMENT NO.	PROJECT NO.
12. PERSONAL AUTHOR(S) See Box 19		TASK NO.	WORK UNIT NO.
TYPE OF REPORT MS Thesis	13b. TIME COVERED FROM _____ TO _____	14. DATE OF REPORT (Yr., Mo., Day) 1985 December	
15. PAGE COUNT 498			
16. SUPPLEMENTARY NOTATION			
17. COSATI CODES		18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)	
FIELD 09	GROUP 05	Local Area Networks, Network Interface, Protocols, UNID, ISO Reference Model, X.25, X.121, TCP/IP	
19. ABSTRACT (Continue on reverse if necessary and identify by block number)			
Title: DEVELOPMENT AND IMPLEMENTATION OF THE X.25 PROTOCOL FOR THE UNIVERSAL NETWORK INTERFACE DEVICE (UNID) II			
Thesis Advisor: Dr. Gary B. Lamont			
Author: Mark W. Weber, BS EE, Capt USAF			
 Lynn Walker 16 JAN 86 AFIT/EN Wright-Patterson AFB OH			
20. DISTRIBUTION/AVAILABILITY OF ABSTRACT  CLASSIFIED/UNLIMITED <input checked="" type="checkbox"/> SAME AS RPT. <input type="checkbox"/> DTIC USERS <input type="checkbox"/>		21. ABSTRACT SECURITY CLASSIFICATION  UNCLASSIFIED	
22a. NAME OF RESPONSIBLE INDIVIDUAL Dr. Gary Lamont		22b. TELEPHONE NUMBER (Include Area Code) 513-255-3576	22c. OFFICE SYMBOL AFIT/EN



Block 19 (cont)

Abstract

This ~~research effort~~ describes the continued development of an improved Universal Network Interface Device (UNID II). The UNID II's architecture was based on a preliminary design project at the Air Force Institute of Technology. The UNID II contains two main hardware modules; a local module for the network layer software and a network module for the data link layer software and physical layer interface. Each module is an independent single board computer (SBC) residing on an Intel multibus chassis, complete with its own memory (EPROM and RAM), serial link interfaces, and multibus interface. The local module is an iSBC 544 and the network module is an iSBC 88/45. The network layer software supports the CCITT X.25, datagram option, protocol and the data link layer software supports the CCITT X.25 LAPB (HDLC) protocol. This report documents the further implementation of the CCITT X.25 protocol in the UNID II design.

END

FILMED

3 - 86

DTIC