THE IMPLEMENTATION OF A LAN

by QUINTIN PETER MCGRATH
B.Sc Eng (Elec), Cape Town

Submitted to the University of Cape Town in partial fulfilment of the requirements for a Masters of Science Degree in Engineering

September 1987
The copyright of this thesis vests in the author. No quotation from it or information derived from it is to be published without full acknowledgement of the source. The thesis is to be used for private study or non-commercial research purposes only.

Published by the University of Cape Town (UCT) in terms of the non-exclusive license granted to UCT by the author.
Abstract

The subject of this thesis concerns the development of a Local Area Network (LAN) for the Department of Electrical and Electronic Engineering at the University of Cape Town.

Motivation for this project was as a result of the ever increasing demands placed on the department's micro-computer training facilities by larger student intakes. The original training system consisted of a PDP 11/23 mini-computer connected via 9600 baud asynchronous links to 11 U.C.T. built micro-computers. This network topology was limiting in three ways:

1. It was slow because of the 9600 baud links and because the PDP was doing a large proportion of the processing.  
2. High-level software development tools for the PDP were too expensive and would over-load the computer. Because the micro-computers have no operating system but only an "in-house" monitor program which is not able to support any high-level language utility, all high-level software tools would have to be individually developed for this particular environment.  
3. Switching was impractical. Because the PDP was the hub of the network all communication between computers had to pass through it. This switching would lead to a greater processing load on the PDP, thus further degrading its performance.

A two pronged attack was used to overcome these weaknesses: firstly, by designing a high-speed (1 Mbps) LAN to provide communications between a PDP 11/23 and up to 30 U.C.T. built micro-computers, faster intercomputer communication as well as switching and resource sharing was facilitated.
Secondly, by customizing an operating system for the micro-computers, standard high-level software development tools could be used on these computers, consequently reducing the PDP’s processing load.

This report details the development of this LAN, beginning first with a study of PDP and the micro-computers, with their respective operating systems. The IEEE 802 standards, a group of standards relating to LANs, are then briefly discussed. A 1 Mbps version of the IEEE 802.4 standard for Token-passing bus networks was used in this project.

The development of the hardware and software to interface both the micro-computers and the PDP 11/23 to the network with the help of a Western Digital access controller chip, the WD2840, is described.

The next level of development was the customization of Digital Research’s CP/M Operating System for use on the micro-computers. Its adaption permitted the use of any CP/M program on the micro-computers, specifically Digital’s CP/NET. This software package creates a network-wide CP/M system, thereby enabling remote program storage, remote print spooling and the use of other remote peripherals.

The final section of the report details the tests performed in order to verify the component parts of the LAN.
Acknowledgements

I acknowledge my heartfelt thanks to the following:

1. My supervisors Prof. H.S. Bradlow and Mr. M. Ventura for their invaluable and untiring help.

2. The Department of Posts and Telecommunications for sponsorship during my masters studies.

3. SANS and MicroCom (Eagle Electric) for the loan of their equipment.

4. Mr. A.S.S.F. da Silva, of SAPONET, for his constructive comments.

5. My wife for her support when things went their worst, and just for being there when needed.

6. JESUS without whom I could not have achieved any of this work. To Him be any glory.
# Contents

1. **INTRODUCTION** ......................................................... 1-1
   
   1.1 History .............................................................. 1-1
   
   1.2 Problems Associated with the Current Network Topology .... 1-2
   
   1.3 Motivation ........................................................... 1-3
   
   1.4 The Proposed LAN ................................................ 1-5
      1.4.1 The Proposal .................................................. 1-5
      1.4.2 Introduction to LANs ....................................... 1-6
      1.4.3 The IEEE 802.4 Standard .................................. 1-7
      1.4.4 The ANSI/IEEE 802.4 Token-passing Architecture .... 1-8
      1.4.5 CP/NET ....................................................... 1-12
      1.4.6 The WD2840 .................................................... 1-13
      1.4.7 The System as a Whole ....................................... 1-13
   
   1.5 Design Status ...................................................... 1-15
   
   1.6 Guide to the Rest of the Report ................................ 1-15
   
2. **CP/M AND RSX-11M** .................................................. 2-1
   
   2.1 Introduction ....................................................... 2-1
   
   2.2 The ISO RM's Application Process ............................... 2-2
<table>
<thead>
<tr>
<th>Contents</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.3 CP/M and the Micro-computers</td>
<td>2-2</td>
</tr>
<tr>
<td>2.3.1 The Micro-computers</td>
<td>2-2</td>
</tr>
<tr>
<td>2.3.2 The CP/M Operating System</td>
<td>2-3</td>
</tr>
<tr>
<td>2.4 RSX-11M and the PDP</td>
<td>2-4</td>
</tr>
<tr>
<td>2.4.1 The PDP 11/23</td>
<td>2-4</td>
</tr>
<tr>
<td>2.4.2 The RSX-11M Operating System</td>
<td>2-4</td>
</tr>
<tr>
<td>2.5 Implementing CP/M on the Micro-computers</td>
<td>2-6</td>
</tr>
<tr>
<td>2.6 The Cold Start Loader</td>
<td>2-7</td>
</tr>
<tr>
<td>2.7 The EPROM Management System</td>
<td>2-7</td>
</tr>
<tr>
<td>3. CP/NET</td>
<td>3-1</td>
</tr>
<tr>
<td>3.1 Introduction</td>
<td>3-1</td>
</tr>
<tr>
<td>3.2 ISO Layers</td>
<td>3-2</td>
</tr>
<tr>
<td>3.3 The CP/NET Package</td>
<td>3-3</td>
</tr>
<tr>
<td>3.3.1 Component Programs</td>
<td>3-3</td>
</tr>
<tr>
<td>3.3.2 CP/NET Utilities</td>
<td>3-4</td>
</tr>
<tr>
<td>3.4 Guidelines for the PDP's Server (Part 1)</td>
<td>3-7a</td>
</tr>
<tr>
<td>3.4.1 Virtual Terminals</td>
<td>3-8</td>
</tr>
<tr>
<td>3.4.2 Drive Allocations</td>
<td>3-9</td>
</tr>
<tr>
<td>3.4.3 Disk Accesses</td>
<td>3-9</td>
</tr>
<tr>
<td>3.4.4 Status and Configurations</td>
<td>3-10</td>
</tr>
<tr>
<td>4. THE LAN INTERFACE HARDWARE</td>
<td>4-1</td>
</tr>
<tr>
<td>4.1 Introduction</td>
<td>4-1</td>
</tr>
</tbody>
</table>
Contents

5.3.4 The Status Report Function .... 5-9
5.3.5 The Isolate Function .......... 5-9

5.4 The Token-bus SNIOS ............ 5-10

5.5 The Requirements of the PDP NIU Software 5-10

5.6 A Description of the PDP NIU Software . 5-11
5.6.1 User Interface and Control Routine (MAIN) .... 5-11
5.6.2 Initialization Routine (INITNET) 5-12
5.6.3 Transmission Control Routine (SNIMSG) .... 5-13
5.6.4 Receive Message Routine (RECMMSG) 5-13
5.6.5 Interrupt Service Routines
(LITCSR and WDSR) .... 5-14

5.7 Guidelines for the PDP Server Program (Part 2) .... 5-15

6. NETWORK VERIFICATION AND CONCLUSIONS .... 6-1

6.1 Network Verification ............. 6-1
6.1.1 Functional Performance ......... 6-1
6.1.2 User Level Software .......... 6-2
6.1.3 Physical Performance .......... 6-3
6.1.4 Data Traffic Performance ....... 6-5
6.1.5 Administration Parameters ....... 6-5
6.1.6 Future Requirements .......... 6-6

6.2 Compliance with the IEEE 802 Standards . 6-7

6.3 Conclusions ................. 6-7
Contents

A. THE BIOS PROGRAM FOR THE U.C.T. MICRO-COMPUTERS A-1

B. THE COLD START LOADER FOR THE U.C.T. MICRO-COMPUTERS B-1

C. THE EPROM MANAGEMENT SYSTEM C-1

D. SABUS LAN CARD CIRCUIT DIAGRAMS D-1

E. THE TAC USER ROUTINES PROGRAM E-1

F. SHILOS FOR THE TAC NETWORK F-1

G. THE PDP 11/23 LAN CARD CIRCUIT DIAGRAM G-1

H. THE PDP LAN CARD TEST SOFTWARE H-1

I. PROGRAM USERS' GUIDE I-1

J. THE U.C.T. LAN ARCHITECTURE J-1
Chapter 1

Introduction

1.1 History

Since its inception in 1980, the computer network within the Department of Electrical and Electronic Engineering at the University of Cape Town has undergone major changes. In that year the department acquired a PDP 11/23, 4 terminals and a printer for undergraduate computer training. The department decided upon two areas of computer tuition:

1. Control or "Real-time" programming
2. Micro-processor assembly language programming

The first of these was provided by RTL/2, a real time language which runs under the PDP's operating system. A masters student, B.G. Sherlock, developed the interface to the control hardware and designed a software simulator. (For further details please refer to reference [1].)

The second area of programming, that of micro-processor assembly language programming, was developed largely by two students, J.A. Eva and J.G. Jack. They contributed to the design and development of a micro-computer and wrote a considerable amount of support software. Their work is described in references [2] and [3].

By the start of the 1984 academic year, which was when this project began, eleven 8085-based micro-computers had been built. Because of cost considerations, no secondary storage was provided on the micro-computers.
Instead, these computers were linked, along with 6 "dumb" terminals and a printer, to separate asynchronous ports on the PDP.

In order to reduce the computing load on the PDP, distributed processing was necessary. A full-screen editor, an 8085 assembler and an assembly code run-time debugging program had been written for the micro-computers. With the aid of these programs, users could create, assemble, debug and execute their assembly code programs on the micro-computers and use the PDP, via the network, for program storage and print spooling. Terminal emulation was also provided, enabling the micro-computers to be used as unintelligent terminals to the PDP for RTL/2 compilation and execution. The micro-computer editor was again used when writing these RTL/2 programs to ensure the least possible load on the PDP 11/23.

Figure 1.1 - The 1984 Network

1.2 Problems Associated with the Current Topology

Three prominent weaknesses of the 1984 network were:

1. Speed - The lines connecting the micro-computers to the PDP operated at 9600 baud. It therefore took approximately 45 seconds to transfer the longest possible file (44k bytes) between the PDP and a micro-computer (excluding the time required for mini-computer disk accesses).
Introduction

- Problems Associated with the Current Topology

2. Switching - In the star network all communications were between the PDP 11/23 and the micro-computers. For two micro-computers to communicate required the invocation of a PDP utility or some program written specially for the purpose. There are in addition a number of computers dispersed around the department which, if linked to this network, could extend the resources available to the students and in turn use the resources provided by the network. The provision of this intercommunication, however, would mean that the already overloaded PDP would have to act as a switch, thereby further reducing its performance.

3. Limited high-level software tools - Because the micro-computers only have a simple monitor program and no operating system, their high-level software tools (e.g. compilers) have to be developed within the department. Similar software for the PDP is too expensive and, if used, would again reduce the mini-computer's performance.

1.3 The Motivation

In the present computer topology, a computer (the PDP 11/23) provides services via a multiplexer unit to a number of terminals and micro-computers. Investigations revealed that with the increase in the number of tasks running on the PDP, terminal response degraded. One of the major tasks performed by the PDP in this environment is the I/O processing. This is clear if one considers that 16 terminal devices, each having its own input and output queues, have to be serviced and that the predominant task is file transfer between the PDP and micro-computers.

A suggested method of reducing this high I/O processing load is to substitute the individual point-to-point 9600 bps links between the PDP and terminal devices with a single high-speed (1 Mbps) multi-drop bus LAN. By using block transfer rather than the slow character-by-character transfer, and because only one port has to be serviced, the amount of I/O processing will be drastically reduced.
A LAN inherently provides a means of multiplexing, thus in a logical sense the original point-to-point system and the LAN system are precisely the same. That is, the PDP 11/23 provides a service to a number of terminal devices attached to it by (logical) point-to-point links. The difference is that the multiplexing function in the latter case is provided by the LAN thus off-loading the PDP.
Once it has been established that a LAN is an acceptable way of improving the teaching system as a whole, the next step was to choose the appropriate LAN architecture. It should be noted that a LAN does not simply define the transmission method and speed, and the access method, but also the protocols used for information transfer, i.e. all the layers of the ISO Reference Model (ISO/DIS 7489) are defined. The choice of the architecture would thus remove both of the above bottlenecks.

The variety of LAN architectures to choose from is almost endless nowadays because of the sudden popularity of office automation and the one-per-desk philosophy. In general, it is best to limit the choice of a LAN architecture to one that complies with an accepted standard, one of which is the IEEE 802 standards. Appendix J contains a lengthy discussion on the different IEEE architectures, their performance, reliability and error handling capabilities. It suffices here to state the conclusions that the author drew based on his discussion in Appendix J:

A version of the IEEE 802.4 Token-passing Bus Standard was chosen because:

1. It has the least complex physical level.
2. The difference in throughput and delay, i.e. the components of performance, between the Token-passing Bus and the other networks was not significant within the requirements of this network.
3. The token-passing Bus is at least as reliable as the other networks defined by the IEEE 802 standards.

Another factor that swung the scales was the availability of an access controlling IC for the token-passing Bus. This chip, Western Digital's WD2840 Token Access Controller, implements a version of the Token-passing Bus architecture which differs slightly from the IEEE 802.4 specification. Where ever possible, i.e. those sections not implemented on
The WD2840, the IEEE 802.4 standard was adhered to. It should be noted that following the IEEE 802.4 standard to the letter would have led to a complex and expensive solution.

1.4 The Proposed LAN

1.4.1 The Proposal

The aim of the project was thus to design and build a 1 Mbps Token-passing Bus LAN incorporating Western Digital's WD2840 Token Access Controller. With the aid of the services provided by the LAN, Digital Research Incorporated's CP/NET had to be used to create a distributed computer system.

From the terminal, the user will simply see a CP/M system. To make the PDP file server and printer facilities available, CP/NET was used. By running CP/NET a user may log onto a number of resource providers or "Servers", as CP/NET calls them, via the network. To allow for this the Token-passing bus architecture was used, as implemented by the WD2840. The network architecture permits the transmission of messages between any two nodes, thus allowing the possibility of more than one Server.

To improve the speed of communications, a 1 Mbps transmission speed was used. (1 Mbps was chosen as it is the maximum speed for the WD2840 access controller chip.)

The construction of this network overcomes the weaknesses of the original network in the following ways:

1. In order to use CP/NET on the network a CP/M environment had to be created on the micro-computers. This meant that the large variety of inexpensive CP/M software became available. At the same time, the processing load was moved from the PDP to the micro-computers.

2. The problem of switching is overcome because the original star network is now replaced by a multi-access bus, which inherently
allows for switching. CP/NET, in turn, provides the software for the switching.

3. Because the network speed is increased to 1 Mbps, the transmission is faster than the combination of eleven 9600 bps lines used in the original network.

This proposal is also expandable and does not require any expensive hardware - two desirable factors.

1.4.2 Introduction to LANs

According to Abrams [4], "a local network is a communications network that provides for the interconnection of a variety of data-communicating devices within a small area." In order to provide a broader view, Tanenbaum [5] lists a LAN's general characteristics as follows:

1. It will usually be small in geographical extent, of the order of a few kilometers.

2. The overall data rates are high - 1 Mbps and above.

3. Both the network and data terminal equipment (DTE) are usually owned by the same company, i.e. it is not a service provided by an external company as is the case in long-haul networks.

Fritz et al. [6] adds to the list:

1. There is usually full interconnection between all the DTEs on the network.

2. A LAN is reliable and has low data error rates - of the order of 1 in 100 million.
3. There is generally no restriction on the type of DTE used on the network.

A LAN therefore has the desired aspects of speed, interconnection and reliability. Furthermore, a LAN is essentially a switch.

1.4.3 The IEEE 802.4 Standard

When embarking on a project of this nature it would be wise to first study the relevant standards, if only to draw out a number of guidelines for development. The IEEE have prepared a suite of standards for LANs known as the 802 standards. For this particular network the IEEE 802.4 standard for Token-passing Bus networks is the relevant work. Below is the background and a brief outline of the IEEE 802.4 standard. (Should the reader require further information he is requested to refer to the ANSI/IEEE 802.4 - 1985 standard.)

With the increased demand for computer communication caused by the development of LSI technology and the subsequent creation of the microprocessor, manufacturers began designing local computer networks based on their interpretation of the ISO RM (ISO/DIS 7498). This led to a proliferation of LAN architectures. In an attempt to standardize on a certain LAN architecture the IEEE appointed a committee, known as the 802 committee.

Very early in its existence the IEEE 802 committee realized that it was unable to standardize on a single LAN architecture. Thus a number of sub-committees were formed to examine the different networks. By the beginning of 1985 standards defining four architectures had been drafted, namely:

1. ANSI/IEEE 802.3 - 1985 defining a baseband CSMA/CD bus network
2. ANSI/IEEE 802.4 - 1985 defining both a broadband and
3. a baseband Token-passing bus network
Introduction
- The Proposed LAN

4. ANSI/IEEE 802.5 - 1985 defining a baseband Token-passing ring network

The ISO RM describes any computer communications system. Those layers that relate to the physical network configuration are the lowest two. It is these layers that the 802 standards define specifically for LANs. The Physical and Data-Link layers are redefined as the Physical (PHY), Media Access Control (MAC) and Logical Link Control (LLC) layers. The PHY layer controls the physical signalling over the network, the MAC layer controls the access to the shared medium, and the LLC layer is in charge of the logical transmission of data over the link.

![Diagram of ISO RM and IEEE 802 Models]

Figure 1.2 - The ISO Model and the IEEE 802 Model

1.4.4 The ANSI/IEEE 802.4 Token-passing Architecture

Token-passing is a distributed polling access method used to provide controlled sharing of a common medium by a number of terminal devices.

Essentially the Token-passing operation is this: the right to the network is passed from station to station around a logical ring on the bus network in the form of a unique control frame called the "token". The station that is addressed by the token may transmit up to a pre-defined maximum number of frames or simply pass the token to its successor. Token rotation is in descending order of station addresses, except that the lowest station passes the token to the station with the highest active address in order to complete the logical ring. Each station keeps a record of its own address, its successor's address and its predecessor's address. By fixing the rotation of the token (to either descending order, as in the
IEEE case, or ascending order which the WD2840 uses), the quick inclusion of new stations and the re-establishment of the logical ring after failure, is facilitated.

![Logical Ring Diagram](image)

**Figure 1.3 - The Logical Ring on a Token-bus Network**

Although the IEEE 802.4 standard defines the lowest two layers of the ISO RM, the whole model holds true as this is a full open system, i.e. a LAN architecture defines all the layers from the Application to the Physical layer.

This method requires much maintenance. Below is a list of the minimum functions that one or more (depending on centralized or distributed control) nodes must perform:

1. **Ring Initialization** - at network start-up or after the logical ring has broken down, the logical ring on the network must be re-established. This requires sorting out who it is who performs the initialization, and who sends the first token.

2. **Station Insertion** - some method of allowing new stations onto an already active network is required.

3. **Station Deletion** - there must also be a method of removing an active station from the logical ring without much disruption of the service.

4. **Duplicate and Lost Token** - a quick and logical method must be provided for the recovery from such conditions.
Introduction

- The Proposed LAN

The IEEE 802.4 standard defines the following method of performing the above functions:

(a) **Node Addition**

Node addition is achieved by the controlled contention process known as "response windows". Each node has the responsibility of searching for a new successor at regular intervals. When the node receives a token, and it is time to search for new successors, it issues a "solicit successor" frame. This invites new nodes with an address that falls between itself and the next node on the logical ring to demand entrance. The token holder will then wait for one response window, i.e. twice the network's end-to-end propagation time. At this point three events may occur:

1. **No response** - meaning that no new nodes are trying to join the logical ring.

2. A "set successor" frame - meaning that only one station wants to join the logical ring. The token holder will then adjust its successor value.

3. A noise - meaning that more than one station in the specified address range wants to be included in the logical ring. The node addition process now enters the second stage, contention resolution. Here the token holder sends a "resolve contention" frame and waits for four response windows. A station that responded to the "solicit successor" frame is permitted to send a second "set successor" frame, but this time it must be delayed by an integral number of response windows, based on the first two bits in the address. Should one of these stations hear another station begin transmitting in a previous window, it must no longer respond. If the token holder received a valid "set successor" frame, it continues as above in 2. If not another "resolve contention" frame is sent. Again, only those stations that responded to the previous frame may respond but this time by delaying transmission based on the next two bits of its
only mode, assuming the worst. A response is dealt with in the same way as the node addition above.

(d) **Logical Ring Initialization**

This process is initiated at network start-up or when one or more stations active on the network detect that the network has been inactive for longer than a predefined time. Once a timer expires, the station sends a "claim token" frame. It then listens to the network, if there is a response, then the contention must be resolved. This is done in a similar way to the node addition process. Here the claimant issues a "claim token" frame at 0, 2, 4 or 6 response windows (or slots - twice the end-to-end propagation time) after the last transmission, this time based on the inverse of the first two bits in the address. It will then listen to the network, if it hears anything it will drop the claim. If nothing is heard it tries again, this time padding the claim by an integral number of slots based on the inverse of the second pair of bits in the address. The station that transmits last assumes that it has the token. The ring is then re-built as described in the node addition section.

1.4.5 **CP/NET**

The popular 8085 operating system, CP/M, created by Digital Research Incorporated, was chosen for the U.C.T.-built micro-computers as they are all 8085 based systems. This proved to be a good choice as, not only is there a vast amount of high-level development software available for CP/M, but Digital Research have also produced a networking package, CP/NET, which enables resource sharing across a network.

CP/NET logically divides the networked computers into two categories, those which provide the facilities, called "Servers", and those which use the resources provided by the Servers, called "Requesters". On the U.C.T. network, the PDP would be the Server and the micro-computers, the Requesters. The micro-computer users would be able to allocate up to 16 logical disk drives, a printer and a console as external or "networked"
devicess. Thus the CP/NET user creates multiple sessions or point-to-point links between his own computer (the Requester) and the one or more Servers. All networked devices are perceived as local by the user, but are accessed across the network by CP/NET.

Digital Research Incorporated supplies CP/NET in assembled code with only the user interface and the Session/Transport Layer interface defined. It is the task of the system designer to provide the lower layers of the ISO RM, i.e. the Transport and Network Layers and those layers defined by the IEEE 802.4 standard. In this network the WD2840 and the author's software and hardware provide the IEEE Layers while the Transport and Network Layers are empty because of the simplicity of this particular LAN.

1.4.6 The WD2840

The WD2840 TAC is a chip which performs the media access control functions on a multi-access bus (as described in section 1.4.4 above). This allows up to 254 computing devices to use the common bus without fear of data loss. The TAC has 16 registers which are used to control the activity of the chip. Thus to use this IC in a network requires hardware and control software to interface it to the physical network and the logical Network Layer in the computer system.

With this hardware and software the WD2840 fulfills the tasks of the three IEEE layers.

1.4.7 The System as a Whole

In the overall sense, the computer network had to look like a CP/NET system. On the micro-computers, firstly, this required the creation of a CP/M environment. To enable the use of CP/M on the micro-computers, the hardware-related portion of CP/M, its BIOS, was modified. The major adaptation was to provide a local "disk drive" on each computer. To achieve this the 128k byte EPROM card contained in the micro-computers,
Introduction
- The Proposed LAN

previously used to store utility programs, was made to look like a CP/M (Read-only) floppy disk. The disk accessing routines in the Basic Input/Output System (BIOS) section of the CP/M operating system were adjusted. The data stored on the EPROM card was also put in the required CP/M disk format.

The CP/NET programs were then run under CP/M. To provide the IEEE layers, the WD2840 and the associated hardware and software was included.

Secondly, the PDP had to be made to look like a CP/NET Server station. In order to use CP/NET on the network, software is required on the PDP to translate the CP/NET requests received from the network into PDP operating system calls and to return the appropriate responses to the Requesters. This section is a project in itself as it requires an in-depth study of the functioning of the PDP’s operating system.

An interface card and the driver software, similar to the micro-computer’s software, was provided.

Diagramatically the system could be described as follows:

![Diagram of the system](image)

Figure 1.4 - The U.C.T. LAN and the ISO RM
1.5 Design Status

Only the part of the LAN designed by the author is complete at this stage. That part consisted of designing the hardware for both the microcomputers and the PDP, writing the software to control the hardware on the micro-computers, and adjusting the CP/M and CP/NET software for the microcomputer's hardware. A test program for the PDP hardware was also written.

Those sections still to be completed by another (unknown) student are the software interface to the PDP's executive (the "device driver"), to enable communication via the network, and the higher layers of Server software on the PDP to provide CP/NET with access to the PDP's resources.

1.6 Guide to the Rest of the Report

Chapter 2 describes the PDP 11/23 and the U.C.T. Micro-computers with their associated operating systems, RSX-11M and CP/M.

Chapter 3 describes the CP/NET software and its adjustment of CP/NET for use on the U.C.T. Micro-computers. A guide to the implementation of it on the PDP is also given here.

Chapter 4 describes the development of the network interface hardware for the micro-computers and the PDP 11/23.

Chapter 5 details the development of the network interface software for the two machines.

The final chapter, chapter 6, provides test results and draws a number of conclusions from the work performed.

The appendices that follow contain the software, the hardware circuit diagrams and a user guide for the network programs.
2.1 Introduction

This chapter deals with the two operating systems, CP/M and RSX-11M. It is in these operating systems that the ISO RM’s Application processes will run.

The purpose of the chapter is to give the reader an overview of the two operating systems. The customization of the CP/M Operating System for use on the U.C.T. micro-computers is also dealt with, as are three utility programs written by the author which enable effective use of CP/M on the micro-computers. Figure 2.1 below shows diagramatically the section of the project about to be discussed.
2.2 The ISO RM's Application Process

The section of the open system model known as the application process is "an element which performs the information processing for a particular application." It can be a manual, computerized or physical process, e.g. an ATM (Automatic Teller Machine) user, an executing program, or a dedicated computer controlling some industrial process.

It is in the operating system environment that the application process will run.

2.3 CP/M and the Micro-computers

2.3.1 The Micro-computers

The micro-computers were designed and built by the students and staff of the Digital Systems Division of the Department of Electrical and Electronic Engineering at U.C.T. Their design is based on the South African Standard Bus [7] (SABus) backplane. The use of this backplane allows the computer to be divided into a number of logical sections, each of which may be implemented on a separate computer card. These cards are then slotted into the SABus backplane.

Each U.C.T. micro-computer contains the following computer cards:

1. An 8085 micro-processor card.
2. A 64k byte RAM card.
3. A card holding 128k bytes of EPROM.
4. A video and keyboard controller card.
5. A communications card providing one parallel and two serial ports.

A "monitor" program, situated in an EPROM on the micro-processor card, enables the user to perform a number of simple tasks which include
examining and setting the contents of memory, adjusting port values, initiating the cpu execution from a specified point in memory and invoking utility programs stored on the EPROM card.

2.3.2 The CP/M Operating System

CP/M is a Disk Operating System (DOS) designed to run on a microcomputer system using either an 8080, 8085 or Z-80 micro-processor as its cpu. A DOS, in a general sense, enables the user to communicate with the disk drives and other peripherals, thereby facilitating reading, storing and running of programs.

CP/M is made up of a number of component programs. The core is the BDOS (Basic Disk Operating System) which controls the logical part of peripheral access. The next section, the Console Command Processor (CCP), has the task of interpreting the command line typed in at the console, recognizing a number of keywords and acting on them. The CCP also initiates the loading of user programs from disk, by means of the facilities provided by the BDOS. The third section of CP/M, BIOS (Basic Input/Output System), is different for each computer system as it is hardware dependent. This section of the software contains the code required to interface to the peripheral hardware. All three programs, the BDOS, BIOS and CCP are loaded into the top of memory. Apart from 255 bytes at the start of memory used for system parameters and temporary storage, the rest of the RAM is available for user programs.

When a program accesses a peripheral it issues a call to the BDOS entry point stating the request number and supplying the appropriate variables. The BDOS then breaks the request into its logical parts, and issues a number of calls to the BIOS. The responses are interpreted by the BDOS and returned to the calling program.

There are a large number of utilities available to the CP/M user, and for a comprehensive list the reader is requested to refer to reference [8]. The CP/M utilities which are used most frequently are:
1. **DIR** - for requesting a directory or catalogue of the programs on a specified disk.
2. **TYPE** - used to print the specified file to the console.
3. **PIP** - a utility which can be used to access the peripherals.

To run an executable file, the user simply types the name of the file, e.g. `SORT<CR>`. This is interpreted by the CCP which instructs the BDOS to load the SORT file. It is then executed. On completion the CCP is again invoked to wait for further instructions.

### 2.4 RSX-llM and the PDP

#### 2.4.1 The PDP 11/23

The PDP 11/23 is a mini-computer which has as its cpu a 16-bit LSI-11 micro-processor. A more complex backplane is used here, namely the Q-Bus [9]. It is more complex in that the backplane simulates a small network using a parallel bus. All transmissions across the bus are supervised by a number of control lines ensuring that data is transmitted correctly.

The U.C.T. computer comprises 128 kbytes of RAM, three disk drives and facilities for 20 asynchronous terminals.

#### 2.4.2 The RSX-llM Operating System

The RSX-llM operating system is a multi-user and multi-tasking operating system. A user may therefore log onto the computer under a particular account, which is isolated from all other users of lower priority, and run one or more programs or utilities. This multi-user, multi-tasking ability is achieved by the time-sharing principle, i.e. each running task is given a fraction of cpu’s time.
The centre of the RSX-11M operating system is the Executive. This is the kernel and is the logic behind the operating system. To interpret the user's command lines, a command line interpreter (CLI) program is used, e.g. MCR or DCL. The executive's interface to the peripherals is provided by the "device drivers", which are dedicated peripheral control programs.

Program access to a peripheral is via a call to the executive, called a QIO. This places a request in the executive's I/O queue. The request will contain such information as the peripheral addressed, the type of request (e.g. read a block of data), the timeout limit, and the internal flag to be set on completion of the request. The executive will initiate the appropriate device driver and order it to perform the task required. Once again the response is interpreted by the executive, and returned to the calling task.

When an asynchronous event is expected, e.g. the input from a terminal, the task could use an asynchronous system trap (AST), which instructs the executive to call the running task when the specified event occurs.

On a well developed system like the PDP 11/23 there are a large number of utilities available to the user. Here are a few that are relevant to the present discussion:

1. HEL - used to log onto the PDP as a user under a specific account.
2. BYE - the utility used to log off the PDP.
3. PIP - is the utility which enables limited control of the peripherals.
4. RUN - is used to begin the execution of a task.
5. SET - may be used to adjust a variety of system parameters relating to the users interface.
6. BRO - allows the user to send messages to one or more terminals.
2.5 Implementing CP/M on the Micro-computers

The RSX-11M Operating System was already running on the PDP but CP/M had to be customized for use on the micro-computers as follows:

The major problem encountered while implementing CP/M on the micro-computers was the lack of a disk drive on the computers. This problem was overcome by making use of the EPROM card, already present in the system, to provide a form of read-only disk drive. The EPROM card previously contained a number of utility programs, e.g. the assembler and the editor.

The card's hardware consists of two address latches, an EPROM counter latch and up to sixteen 8k by 8 bit EPROMs (2764s) with the appropriate buffering. To access the data, the address and EPROM number are written to their respective latches after which data is read.

Because the micro-computers may be used as stand-alone computers, the utility programs must be available should the operating system not be loaded. To enable this the start of the "disk" on the card is uniquely marked. When the operating system is loaded the cold start loader locates the start of the "disk" and informs the BIOS routines. These routines then apply the appropriate offset when accessing the EPROM card.

The method used for program storage in the stand-alone system is different to the CP/M disk storage. The former uses two unique bytes to indicate the start of file. Following this marker is the program name and its length. The contents of the file then follow. CP/M, on the other hand, uses a directory which contains a list of the programs stored on the disk and the number and position of the sectors containing the programs. Thus to access a file each sector position must be calculated. Converting from one method to the other is clearly not practical. Figure 2.2 below shows a map of the two methods. A combination of the two methods can be used on condition that the CP/M "disk" follows the stand-alone storage.
The customized version of the BIOS may be found in appendix A of this report.

2.6 The Cold Start Loader

On station start-up there must be some way of loading the operating system into memory and initiating execution. This is the task of the Cold Start Loader program, a listing of which may be found in appendix B. It writes the three DOS component (BDOS, BIOS and CCP) programs into memory and finds the start of the "disk" on the EPROM card. The cold start program then initiates the DOS by calling the BIOS Cold Boot routine. This routine initializes the system parameters and then executes the Autoload command line (in this case CP/NET is loaded). By setting this line to contain nulls the micro-computer can be used as a stand-alone CP/M system, i.e. CP/NET will not be loaded.

2.7 The EPROM Management System

EMS, the EPROM Management System program, enables manipulation of the programs stored on the EPROM card. It is a short PASCAL-MT+ program which runs on a standard CP/M computer that has at least one disk drive. (A listing of EMS may be found in appendix C.)

On a standard CP/M system, each program stored on disk is broken up into 128 byte blocks and written to (128 byte) sectors. The position of these sectors is recorded in the directory, along with the file name and a count of the number of sectors used. The directory entries are stored sequentially in the first sector of the disk.

To add a new file to the EPROM disk, EMS reads the specified file from disk into RAM, breaks it up into the required 128 byte blocks and then writes it to a number of files on disk. These files, each a maximum of 8 kbytes long, define the data that must be contained in each EPROM of the
EPROM card. The entry is then generated and added to the directory contained in the "directory" EPROM file.

To delete a program, EMS simply adjusts the directory EPROM file to exclude the specified file. This is done by moving all the entries that follow the deleted file up by one entry position.

The data contained in these EPROM files generated by EMS may be used to program the EPROMs which are then inserted into the EPROM card.
3.1 Introduction

This chapter provides an overview of CP/NET and its working. This package consists of two sections, the commercially supplied software, NDOS, which is the interface to the CP/M software, and SNIOS, the software that provides the link between NDOS and the network. This chapter is mainly concerned with NDOS, although CP/NET as a whole is also dealt with. Because NDOS is not adjustable, it was thus used as is on the micro-computers. The PDP uses a different operating system, so the PDP version of NDOS has to be written from scratch. This program, called the SERVER, is a task running on the PDP which will provide the user access to the network. It will call the operating system executive to initiate NETWRKIF, the LAN device driver, to interact over the network, and use inter-task communication to converse with the user/application task. Guidelines are provided herein to aid the student who is to embark on this project.

The sections of the project being discussed in this chapter are:
Figure 3.1 - Chapter Topic
3.2 ISO Layers

The three layers being considered in this chapter are those that provide the Application Process (AP) with an interface to the network. The lower layers are concerned rather with the actual transmission of the information across the network.

Briefly, the Application Layer is the AP's window to the network, and so provides a means of communicating with other APs attached to the network. It has to transform the data into a format that can be sent over the network. In some networks where distributed processing is performed, the Application Layer has the task of controlling the resource usage. The AP/Application Layer interface simply defines the transmission and reception of data.

The Presentation Layer is responsible for insuring that the format of the information is acceptable to the Application Layer, and thus the AP. Some examples of Presentation Layer functions are syntax changes (e.g. the conversion between ASCII and EBCDIC), encryption and decryption (to provide secrecy) and data formatting (e.g. space and character compression). The interface to the Application Layer defines the transmission of Application Data Units as well as the parameters to select the required presentation functions.

The Session Layer manages the connection between two presentation entities in that it is in charge of the connection and disconnection of the link (or "session"). This requires the transformation of an AP address into a Session Layer address. It is also in control of the dialogue, i.e. whose turn it is to transmit next and the speed at which data is presented for transmission. The Presentation/Session Layer interface defines the transmission of Presentation Data Units, synchronization control and parameters for exception reporting.
The NDOS and SERVER programs perform the services required by these three layers. Note that some of the ISO RM's services are not performed as they are not required in this environment.
3.3 The CP/NET Package

3.3.1 Component Programs

When a micro-computer takes part in a network it loads the CP/NET programs, namely, NDOS and SNIOS. The NDOS (Networked Disk Operating System) is the user interface part of the package, i.e. it performs the Application, Presentation and Session Layer functions and provides their services. The SNIOS (Slave Network Input/Output System) performs the network oriented services, i.e. the rest of the ISO RM services. SNIOS is therefore entirely network dependent while NDOS is network independent.

In order to achieve network access the NDOS intercepts all peripheral calls to establish whether they are for local or networked devices. If a call is for a local device it is passed to the BDOS and the standard CP/M access continues, i.e. the BIOS is instructed to perform the required function, say, read from disk. Responses are returned to and interpreted by the BDOS, and the results are returned to the calling program. If, on the other hand, the peripheral call is for a networked device, the NDOS creates a packet and sends it across the network to the Server station using the network interface provided by the SNIOS program.

On the Server side, the packet received from the Requester is interpreted and the appropriate operating system calls are issued. When a response is received, it is relayed to the Requester via the network. This whole task is achieved by two programs, the network interface program (NETWRKIF), which performs the same functions as the Requester's SNIOS, and an operating system interface program (SERVER).

Because a number of Requesters may use the same Server and each of them requires protection from other users, a multi-user operating system must be used. On the standard CP/NET network the Server computers will run another Digital Research product, MP/M-II, a multi-user version of CP/M. On the U.C.T. network, the PDP 11/23 will be the only Server at this stage, and so the tasks normally performed by the MP/M-II Server will be performed.
by a software package (still to be completed by another student) running under RSX-11M, the PDP's operating system.

3.3.2 **CP/NET Utilities**

The features of CP/NET fall into three categories:

1. Networked peripherals
2. Inter-computer message transmission
3. Electronic Mailing

The extra facilities available to the CP/NET user are initiated by typing the appropriate keyword. These are recognised by the extended console command processor program, CCP, and are:

1. **CP/NET Loader** - This is the utility used to load the CP/NET programs and to relocate and extend the CP/M Operating System programs.

2. **Logon** - enables the CP/NET user to attach the CP/NET computer to a Server computer as a Requester. A Server may have as many as 16 Requesters logged on at the same time. The Requesters may similarly, be logged onto up to 16 Servers.

3. **Logoff** - detaches the Requester from a specified Server.

4. **Network and Local** - allows the user to assign a peripheral to an attached Server, and to re-assign them as local. 16 disk drives, a printer and a console may be assigned as networked devices.

5. **Control-P and Endlist** - will begin and end an echo of the console display to the printer which may be local or remote.

6. **Disk Reset** - allows the user to reset the parameters of a remote disk. This is required when a disk is changed on an MP/M computer, and has
the same effect as a Control-C on a CP/M computer, i.e. it reads the directory track to ensure that subsequent access to the new disk is correct.

7. Status - will display a table of peripherals indicating which are local and which are networked.

8. Mail - initiates the menu-driven Electronic Mailing utility.

Under CP/M, the user may access BDOS directly via an assembly code program. For example, the user may reset the computer system, write characters to the console and so on. NDOS may be accessed in the same way. This is facilitated by defining 14 "function codes" which are appended to the standard CP/M function codes.

To access the network in this way, which may be thought of as a second AP/Application Layer interface, the user sets up the parameters as shown in table 3.1 and calls the NDOS entry point at 0005H. The NDOS program will perform the function and return to the calling process. (The table below gives a list of the functions available; should more information be required please refer to reference [9].)

<table>
<thead>
<tr>
<th>Code</th>
<th>Function</th>
<th>Input</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>38</td>
<td>Access Drive</td>
<td>DE: Drive vector</td>
<td>none</td>
</tr>
<tr>
<td>39</td>
<td>Free Drive</td>
<td>DE: Drive vector</td>
<td>none</td>
</tr>
<tr>
<td>42</td>
<td>Lock Record</td>
<td>DE: FCB address</td>
<td>A: Error code</td>
</tr>
</tbody>
</table>
Table 3.1 - NDOS Function
The CP/NET Package

At the other end of NDOS, the information is very similar except that the source and destination addresses (each 8 bits long) have been added, as well as a format control code, fixing the lengths of the fields of the buffer, and a size field, indicating the number of Session data units are in the buffer. The message buffer format is (each field is one octet long):

1. FMT - the format control field, containing a 0 for the request message and a 1 for a response from the Server.

2. DID - the destination ID field. It may have a value of 1 to 255.

3. SID - the source ID field, i.e. the address of this station. Again, the value range is 1 to 255.

4. FNC - is the function code and will be one of the C values in table 3.1, or any of the CP/M codes applicable to peripherals.

5. SIZ - gives a count of the number of message octets in the buffer. It has a range of 0 to 255, and is one less than the number of MSG octets.

6. MSG0 to MSGn - are the message octets, where n is SIZ plus one. The values of these octets is 0 to 255.

The DID and the FNC together define the particular application being addressed on the server.

It is this message buffer that must be sent across the network by SNIOS and the responses from the Server must be returned to the NDOS program in the same format.

Relating the NDOS functions to the services of the top 3 layers of the ISO RM now:
The user interface to the network (or the application layer) is provided via the function calls (specified in table 3.1) or via the utility programs.

The NDOS and SERVER programs convert the function calls into a message containing the relevant address, length and format fields.

The session is established and disconnected by the LOGIN and LOGOFF commands received either via function calls or by the utility program. The NDOS/SERVER interaction works on a strict command-then-response routine thereby defining the sequence of transmissions.

3.4 Guidelines for the PDP's Server (Part 1)

There are two distinct parts to the PDP's network package, the SERVER program which will be the interface to the PDP's operating system, and the device driver program, previously called NETWRKIF. In this section only the guidelines for the SERVER program are given, once the WD2840 control
software has been discussed, the NETWKIF program guidelines will be given (see Chapter 5).

In the standard SERVER program in the MP/M II Server station, each Requester that logs on is given its own SERVER process. Thus as the Requester logs in the SERVER generates a new task to deal solely with the new Requester's requests. The limit of 16 Requesters at one time will define the upper limit of the memory requirement. It is suggested that the PDP's SERVER process follows the same sequence.

3.4.1 Virtual Terminals

The first task is convince the RSX-11M's executive to allow a multiple number of users to be logged onto the same physical peripheral simultaneously. There will be only one physical input from the network to the PDP. One suggestion is to use the "virtual terminal" facility provided by RSX-11M. This allows the user to generate a number of logical terminals linked to a task with the real terminal as the port for input and output. Thus there will be a multiplexing/demultiplexing task which is the first to get the message from the device driver. It will then select the correct SERVER process and pass the request on. In this way each Requester could log onto the PDP via its own virtual terminal.

The creation of a virtual terminal is a simple matter via the executive's Create Virtual Terminal function. It is possible to initiate execution of a process from a second task by means of the Spawn function. Using the Spawn function in the multiplexer/demultiplexer task would enable it to start a new SERVER process for each Requester. At the termination of the session, i.e. when the Requester Logs off the Server, the parent task could be informed and the virtual terminal deleted.
3.4.2 Drive Allocations

CP/NET allows the allocation of 16 disk drives on one or more Servers. Generally, when a user signs onto the PDP from an unintelligent terminal he is assigned a number of defaults, namely an account number, a disk drive and a terminal number. From within RSX-11M a user is able to change these defaults, so is able to access a drive other than his default drive. It should thus not be difficult to allow the user to define a particular drive (i.e. A: through P:) on the CP/NET computer as a networked on the PDP (i.e. DL0: through DL2:) and an account number, or as the magnetic tape driver, MNO:. For example, B: may be networked as DL2:[1,20], i.e disk drive 2 and account number 1,20.

A problem is that CP/NET simply uses a vector 16 bits long to indicate which drive on the Server is being selected. Because the NDOS program is unalterable (or at least without much effort), a selection of defaults would have to be set up before networking a device so that a single bit in the vector would have meaning. The vector would therefore have to either be predefined on the Server, or be set up by a program run when the user logs on. Because most of the students simply use their default account and drive, a predefined vector may well be the best solution, e.g.:

Drive A: on the server (bit 1 in the vector) is the default account and drive.
B: (bit 2) is the default account and drive 0.
C: (bit 3) is the default account and drive 1.
D: (bit 4) is the default account and drive 2.
E: (bit 5) is the default account and the tape drive.
and so on ....

3.4.3 Disk Accesses

A CP/M disk uses a 128 octet block structure, and so all access are in 128 octet blocks. This is allowed by RSX-11M although it may be more
efficient to access larger blocks. This can be done assuming that subsequent blocks are going to be requested.

More complex is the directory structure and the file control information. It would be unwise to duplicate RSX-11M's directory information and so some translation is preferable.

The file control abilities under RSX-11M are considerably more powerful than CP/M's and so the required file control facilities will easily be supplied, although RSX-11M's abilities in this field will not be used to the full effect. The required facilities include record locking (to allow access to only one user), allowing two tasks to update the same file and file appending.

3.4.4 Status and Configurations

There are a number of configuration and status tables that must be supplied by the Server on request. Most of these tables bear no relation to the PDP and so will have to be generated in the SERVER process. These tables include the configuration of the Server, i.e. which drives (A: through P:) are being used by which Requester, and the status of the network, which would have to be calculated by the SERVER process on the basis of information supplied by the NETWKIF process.
Chapter 4

The LAN Interface Hardware

4.1 Introduction

This chapter is the first of two chapters dealing with the network interface layers of the LAN. This chapter jumps to the PHY and MAC layers of the architecture, while the following chapter deals with the LLC and Network layers. In this chapter the WD2840 Token Access Controller (TAC) and the hardware required to drive it are described, i.e. the hardware that forms the bridge between the physical network and the computer's backplane. The next chapter will detail the software required to interface it to the CP/NET programs. In this way all the network interface programs of the ISO RM (the Transport and Network Layers) and the IEEE layers which replace the Data Link and Physical Layers, i.e. the LLC, MAC and PHY Layers, will be implemented.

![Diagram of network layers and hardware]

Figure 4.1 - Chapter Topic
The first section of the chapter provides the information about the micro-computer and PDP backplanes. The WD2840 from a hardware point of view is then described. Finally, the interface hardware for both the micro-computers and the PDP is described.

4.2 The ISO RM Network Related Layers

There are four layers that make up the network related layers in the ISO RM, the Transport, Network, Data Link and Physical Layers. In the LAN environment the Data Link and Physical Layers are replaced by the three IEEE layers, the Logical Link Control (LLC), the Media Access Control (MAC) and Physical (PHY) Layers.

The Transport Layer is the end-to-end network control layer and so supplies the Session Layer (and thus the higher layers and AP) with an error-free link. It will generally break the message received from the Session Layer into small blocks, and concatenate the parts of the Session data unit received from the Network Layer to make one Session data unit, or perhaps vice versa. This calls for the ability to sequence the blocks of data so that the information received is the same as the sent data. It is responsible for the end-to-end flow control and error recovery. In larger networks it may use a number of Network Layer links between the same two Transport end points, these must be controlled and addressed properly.

The Network Layer is responsible for routing the data through the network, which, in large networks, requires a number of intermediate point-to-point links. The short point-to-point links must be flow controlled and checked for errors by the Network Layer. It must be capable of resetting the network links.

The LLC Layer is responsible for the link control functions of the Data Link Layer, i.e. the delimitation and synchronization of the transmission, the sequence control over the link and any error recovery that is required once an error is detected.
The MAC Layer has the task of controlling the access to the common medium. It must thus provide access facilities, address recognition, error detection facilities and data transmission.

The PHY layer is responsible for the bit-by-bit transmission and reception of data. It must perform unit framing and frequency or amplitude conversions, e.g. Manchester Encoding/Decoding and FSK Transmission.

4.3 The U.C.T. Micro-computer's Backplane

The U.C.T. Micro-computers use the SABus [7] backplane. This backplane consists of 64 lines. Below is a description of those relevant to this project.

The first group is the 8 bi-directional data lines. The drivers for these lines must be tri-stated (i.e. in high-impedance state) when they are not driving the lines. There are 16 address lines which are again bi-directional. This is the second group of lines which, once again, must be tri-stated when not in use.

The third group is made up of a variety of control lines, those relevant to the discussion are:

1. Memory Read (MR) and Memory Write (MW), which are used by the controlling device (the cpu or a DMA controller) to activate the micro-computer's memory.

2. Output Write (OW) and Input Read (IR) are two lines used by the controlling device to receive input from and give output to an I/O device.

3. The WAIT line is an open collector driven line which is used by the addressed device to inform the controlling device that it is not yet ready for the data. This line is most commonly used by the RAM to
indicate to the cpu that it is not ready for data or that the data on the backplane is not valid.

4. HOLD and Hold Acknowledge (HOLDA) are two lines connected to the cpu which enable the use of more than one controller on the backplane, e.g. a DMA controller will activate the HOLD line to indicate that it has information to pass to the RAM. Once the cpu has completed its present cycle, it relinquishes the backplane and indicates to the DMA controller via the HOLDA line. The DMA controller performs its transfer and de-activates the HOLD line allowing the cpu to continue as before.

5. RESET is a line, activated by the cpu, which initiates the resetting of all the cards attached to the backplane.

All the levels on the SABus backplane are TTL levels. The different timing diagrams for the relevant read and write cycles may be found in Appendix D.

4.4 The PDP's Backplane

The PDP's Q-Bus [8] backplane is similar to the SABus although it has much stricter specifications for the attached devices and their voltage levels because it is run at a higher speed. All devices attached to the backplane must use open collector drivers, with high-impedance input buffers. Below is a description of the relevant lines.

The 16 address/data lines (BDAL0 - BDAL15) are multiplexed under control of the synchronization line (BSYNC) and the two input/output control lines BDIN and BDOUT. The write cycle line (BWTBT) is also used during the addressing phase to indicate if a write or read cycle is to follow and whether it is a byte or word long. The addressed device indicates acceptance of the data (for a write) or the stability of the data (for a read cycle) via the reply line (BRPLY).
In the Q-Bus, 128 kbytes of memory may be addressed, but the top 8 kbytes are set aside for the I/O page. Thus to reduce the decoding in I/O devices a control line (BBS7) is activated each time the address is in the I/O page.

The PDP's interrupt structure is designed to enable both priority levels and, within each priority level, position dependence. The high priority devices are serviced first. If there are a number of devices with the same priority interrupting simultaneously, they are serviced in the order of their proximity to the cpu. This structure is illustrated below in Figure 4.2.

![Figure 4.2 - PDP Interrupt Structure](image-url)

Under interrupt conditions the device will activate the interrupt lines (BIRQ4 to BIRQ7) which gives the priority of the interrupt. Once a response is received from the cpu, via the BIAKI line, the vector is given on the BDAL lines under control of the BDIN and BRPLY lines.

To initialize the devices attached to the backplane the cpu activates the BINIT line.

The timing diagrams for the read, write and interrupt cycles may be found in appendix G.
4.5 The WD2840 Hardware Interfaces

This section describes the three interfaces on the WD2840 TAC chip. These interfaces are a DMA interface, a network interface and an interface to the 16 host-accessible status/control registers on the TAC. The WD2840 timing diagrams may be found in appendix D.

4.5.1 The DMA Interface

The DMA interface permits the transfer of frames and frame related counters between the WD2840 and RAM. The frame related counters give such information as the number of frames received, the number of transmission retries and the number of erroneous frames received.

This interface consists of 16 address lines (A0-15) and 8 data lines (DAL0-7). The TAC provides two DMA request lines (DRQ0 and DRQI) and a DMA request granted line (DACK) which are used to control the DMA access and provide the RAM control lines during DMA.

4.5.2 The Network Interface

The network interface provides the signals required for a Non-Return to Zero Inverted (NRZI) modem interface. The WD2840 uses synchronous transmission and so provides lines for both the receive and transmit clocks (RC and TC) and their respective data lines (RD and TD). The other network interface lines are request-to-send (RTS), clear-to-send (CTS) and signal quality (SQ).

4.5.3 The Control Interface

The WD2840's activity is controlled by 16 host-accessible status/control registers which may be accessed via the third interface. These registers provide such status information as the reason for the
interrupt, whether the previous command is complete and present successor's address. The host computer, on the other hand, will provide commands like attempt to enter the logical ring, accept new successor address and transmitter or receiver enabled.

This interface consists of four address lines (IA0-3), eight data lines (DAL0-7) and the three control lines, being the chip select (CS), read enable (RE) and write enable (WE).

4.6 Design Philosophy

The design of both of the NIU cards (and the control software described in the next chapter) employed Julius Caesar’s philosophy of "divide and conquer". This was achieved by breaking down the task into a number of logical blocks which were developed and debugged as separate entities. Further, their development followed the steps given by Short [10] for the design of a micro-processor based system, shown diagramatically in figure 4.3 below.
From the references [11,12,13], it is evident that a stand-alone NIU is a popular implementation. This method was not employed here as it would create a separate unit that would require its own power supply, making it both costly and bulky. Secondly, because of the 1 Mbps operating speed, an additional high speed interface to the computers would have to be designed.

Instead, two compact computer cards were developed, one to slot into the micro-computer's backplane and the other into the PDP's Q-bus.

These circuits were initially designed using logic gates and prototyped on wire-wrap board to verify their functioning. Once fully tested, a new version, incorporating the logic into Programmed Array Logic chips (PALs), was designed and tested for each NIU. The PALed designs were then transferred to printed circuit boards.
PALs were used in the final design instead of logic gates. This was for three reasons: their speed, their cost and their size. With regard to speed, the PAL is able to realize even complex functions within 20 nanosec, whereas the discrete solution's relation to time is determined by the number of gates used.

Secondly, on average a PAL replaces between 4 and 6 logic gate ICs but only costs in the order of 3 times that of a single logic gate chip.

One PAL is also much smaller than 5 or 6 logic gates, thus space is saved on the printed circuit board. A certain degree of flexibility with regard to the input and output pins on the PALs also allows for more efficient printed circuit board layouts.

4.7 The Micro-computer LAN Card

4.7.1 Design Description

Below, in figure 4.4, is a block diagram of the Micro-computer interface card.

(a) The DMA Interface

The WD2840 is a real-time device which requires immediate access to the RAM where the data frames and control information is stored. One possible method of ensuring this access is to provide it with a block of local RAM which could be accessed at will. This RAM would be separate from the host's RAM and so would not be affected by host activity. The computer's CPU would only have access to this memory when the WD2840 was not using it. A second possible method is to set aside a part of the host's RAM for use by the WD2840. The WD2840 DMA control facilities could then be used to control RAM access.
Selection Circuits
Scan Control
Read and Write enable
Token Access Controller
Network Control and Interface
Clock Generation Circuits
Data
Data and Control
Clock
Chip Select
Data/Addr
Data/Addr
Data/Addr
Data/Addr
Ctrl
Line Buffer
Ctrl
Line Buffer
DMA Control Circuits
DMA Request
Address
Address
The first method above permits the use of a number of real-time devices, such as disk drives and asynchronous terminals, because the NIU can continue to access its local memory without holding up the rest of the computer's activity. When the NIU requires service, e.g. frame reception complete, it can flag the CPU which will service the interrupt when it has time.

The PDP's NIU used this method because of the other real-time devices attached to the computer which could not be stopped at will. In the case of the micro-computers, on the other hand, no other real-time devices are attached to the computer and so part of the host's RAM was set aside for NIU use. This also allowed for a cheaper solution.

In the micro-computer, bus (and therefore RAM) arbitration is achieved by activating the SABus HOLD line under control of the WD2840 DMA request lines. This bus line is connected to the 8085's /HOLD input, which when asserted, instructs the 8085 to complete its present cycle and then relinquish the bus. Once the bus is available the WD2840 is informed via the SABus hold acknowledge (/HOLDA) line. The WD2840 then uses its DMA request lines to perform memory reads and writes by activating the SABus Memory Read (/MR) and Memory Write (/MW) control lines. Dynamic RAM is used in the micro-computers and so the NIU has to accept short delays whenever the RAM requires refreshing.

The TAC address and data lines are buffered to provide isolation and prevent overdriving the WD2840's outputs.

(b) The Control Interface

The second section of the WD2840's host computer interface allows the host CPU to control the functioning of the TAC. This is achieved by adjusting the values of 16 status/control registers on the WD2840. There are two limitations on the access of these registers - they may not be accessed while the TAC is performing a DMA transfer and there is a minimum
inter-access time that must be upheld. The first of these limitations cannot occur in this implementation as the cpu is in hold state.

Should the cpu attempt to access the TAC during the inter-access time, the SABus WAIT line is activated until the TAC is ready. The assertion of the WAIT line causes the cpu to enter an idle state.

Access to the TAC registers is gained by activating the Read Enable (/RE), Write Enable (/WE) and Chip Select (/CS) lines on the WD2840. The /RE and /WE pins are connected to the Input Read (/IR) and Output Write (/OW) lines on the SABus backplane. The four most significant lines of the 8 address lines are used to select the card and thus the TAC. The position of this card in the 8085's I/O page is chosen by selecting a 16 byte boundary address via mini-jumpers on the interface card. The four least significant address lines are used to select one of the 16 TAC registers. The data lines are buffered through bi-directional, tri-stateable buffers controlled by a combination of the /IR, /OW and the card select lines.

(c) The Network Interface

The WD2840 has the necessary signals to interface it to a NRZI modem. The modem has to provide both the receive and transmit clocks, the required signal levels for the network as well as network isolation.

The IEEE 802.4 standard requires Manchester encoding when Phase-continuous FSK is used. Since this PHY layer may be used on the U.C.T. LAN in the future (once the Signetics FSK chip pair becomes available), manchester encoding was included. This method also provides a quick and reliable method of clock recovery from the transmitted data.

The encoding is performed in this LAN by a Harris Digital IC, the HD-6409. This chip provides a fixed frequency Transmit clock and derives a Receive clock from the received data. By connecting this chip to the TAC, data may be clocked into the HD-6409 (from the TAC's TD pin) on the rising edge of its Encode Clock (ECLK). The TAC uses the inverted image of the
encoder's Decode Clock (DCLK) to synchronize to the data stream being received (on its RD pin) from the HD-6409.

The Manchester Encoder also provides an output to indicate whether the data being received is valid. This output is connected via an inverter to the Signal Quality (/SQ) input of the WD2840. If, at any stage during the frame reception, the encoder discovers an invalid Manchester character, the Signal Quality line is deactivated and the TAC discards the frame.

The encoder's 16 MHz clock input is supplied by a passive crystal oscillator circuit. As an output from the encoder there is a 16 MHz TTL square wave which is used to drive both a divide-by-8 counter for the WD2840's clock input, and the timers for the synchronous circuitry.

On the network side of the HD-6409, an RS-422 receiver/driver IC drives the twisted-pair cable directly. This chip is enabled by an inverted image of the TAC's RTS line. Its task is to provide the correct signal levels and network isolation as defined by the EIA RS-422 standard.

(d) The Solicit Timer

The TAC can use either centralized or distributed solicit control. The vulnerability introduced by a centralized controller led to the choice of distributed control for this implementation. This choice also complied with the IEEE 802.4 standard.

The solicit timer, which is part of the MAC layer, defines when the station must search for a new successor. As there is no real-time clock on the computer, which would generally be used to define the inter-solicit period, a solicit timer had to be included in the hardware. This timer was not implemented in software due to the variety of programs that will be run.

At regular intervals (determined by this hardware timer) the TAC is instructed to scan until a (new) successor is found. This is achieved in
hardware as follows: The value of the CR1 register (which controls the successor address value) is read into an octal flip-flop. The required bit is set and the new value written back to the TAC’s CR1 register. The TAC will then set its successor’s address to the value given, which is pre-set to its own address plus one at start-up. The timing is provided by a PAL clocked by the 16 MHz signal from the Manchester Encoder. To address the CR1 register, a buffer which has its inputs hardwired to the correct value, is selected. Should the host cpu attempt to read the registers during this period, the SABus WAIT control line will be asserted until the end of the cycle.

(e) Reset Circuitry

The reset line from SABus backplane is used to reset both the TAC and the Manchester Encoder. A 10 msec reset pulse is supplied by the cpu via the backplane, thus no extra circuitry is required apart from the usual buffering as specified by the SABus Standard.

A full circuit diagram of the micro-computer NIU, the PAL equations and timing diagrams are given in appendix D.

4.8 The PDP Interface Card

4.8.1 Problem Description

The PDP’s NIU must be designed to work with a number of other real-time devices, namely 3 disk drives, a tape drive and asynchronous terminals. The tape drive, disk drives and some of the terminals use the PDP’s DMA facility and so it must be able to access the RAM at any time and without extreme delays.

Two possible choices for the TAC’s DMA interface to the PDP are available: The first is to use a high priority DMA channel to transfer the frames to and from the system RAM; the second is to use some form of local
The LAN Interface Hardware - The PDP Interface Card

RAM situated on the interface card which can be accessed by both the TAC and the PDP 11/23 cpu.

The use of the local (or dual-port) RAM was prompted by the TAC's critical timing requirement and because there is no way of prioritizing the PDP's DMA transfer. Thus if another device was in the process of performing a DMA transfer, the TAC would have to wait for completion of the DMA cycle before it could use the bus. The dual-port RAM, on the other hand, allows the WD2840 to continue as a separate entity until a significant event occurs, such as token received, data frame received and frame sent. It then interrupts the PDP's cpu for service. (The WD2840 conveniently provides an interrupt line that can be used to generate this interrupt.)

A problem with dual-port RAM is that there is limited space on the I/O page - 8k bytes - which cannot all be dedicated to the LAN controller. Some means of mapping the RAM on the TAC board into a small section of the PDP's I/O page had thus to be found.

A second problem, related to the I/O page, was the accessing of the TAC's 16 status/control registers. As all the registers in the I/O page have to be on a word boundary, the 16 registers would require 16 words of I/O page which is excessive.

To reduce the I/O page address space requirement, the NIU is accessed via four registers. The first two are used to access the RAM while the second pair enable access to the TAC's status/control registers.
4.8.2 Description of the Implementation

A block diagram of the circuit is shown below. A detailed circuit diagram and the PAL logic functions may be found in appendix G.

The relevant Q-Bus timing diagrams are also given in appendix G of this report.

(a) The Control Interface

To provide versatility, the exact position in the PDP's I/O page of the NIU card, and therefore the 4 registers used to access the RAM and the TAC's registers, is mini-jumper selectable on the interface card.

Access to the TAC's registers is achieved by writing the address of the register to be accessed into the first of the four registers. This address is latched into a flip-flop until a new value is written to it. The addressed WD2840 register may then be read from or written to. In doing so the correct control lines, Chip Select (/CS) and Read Enable (/RE) or Write Enable (/WE), are activated under PAL control.

A shift register is used to enable the correct timing and to ensure that the WD2840 is not accessed more often than permitted, i.e. with less than 300 nanosec between chip selects.

(b) RAM Arbitration Logic

Some form of RAM arbitration had to be provided to enable the use of the RAM by both the PDP cpu and the TAC. This was done on a byte-by-byte, first-come-first-served basis with the TAC having the precedence should the two machines attempt to access the RAM at the same time.
The LAN Interface Hardware
- The PDP Interface Card
Because the RAM arbitration is done on a byte-by-byte basis, the TAC access is never delayed for an unacceptable period while the PDP's cpu accesses the RAM.

(c) Accessing the RAM

From the WD2840 side: once the arbitration logic circuitry has granted permission, the WD2840 RAM address buffers are enabled simultaneously disabling the PDP's buffers. The arbitration logic then sets a multiplexer to derive the RAM control lines (Output Enable, /OE, and Memory Write, /MW) from the WD2840 and not from the PDP's bus. The address is set up by the WD2840 via latches to ensure that it remains valid for the time required by the RAM. The chip selection of one of the four 2 kbyte RAMs is via a two-to-four line decoder, where address lines A11 and A12 are provided as inputs. During a write operation the data lines are latched to ensure that the data is maintained long enough for the RAM. The overall timing is controlled by a PAL.

From the PDP's cpu side: One method of reducing the I/O page requirement is to write the address to one register and then to read from or write to another register which simultaneously enables the address in the first, as in the case of the status/control register access above. But when the cpu accesses the RAM, it will generally access data in sequential addresses. To improve the efficiency, the "address register" is replaced by a 16-bit loadable counter. The cpu will thus first set up the starting address of the block of data by writing it to the address counter. The data may then be accessed via the second register without having to adjust the address register each time, as long as the access is sequential. A hardware requirement is that the address is one less than the first byte to be accessed as the counter is incremented at the start of each cycle.

Some versions of the PDP's cpu, specifically the version in the computer used to prototype the NIU card, use a read-modify-write cycle for their write cycle. To ensure that the address counter was not incremented twice, the top address line, A15, was set or cleared to indicate whether the following cycle was a read or a write respectively.
The LAN Interface Hardware
- The PDP Interface Card

Once the arbitration logic had granted the cpu permission to access the RAM, the RAM was accessed as described above for the WD2840. The control lines to the RAM and the bus response lines are controlled by a PAL.

(d) Interrupt Circuitry

The WD2840 supplies an interrupt pulse whenever a significant event occurs. This pulse is latched onto the bus until the cpu responds. The priority of the interrupt is mini-jumper selectable as is the interrupt's vector address. The required NIU responses are under PAL control.

(e) Bus Interface Circuitry

The bus interface section of the circuitry is somewhat unusual because the bus specification requires high impedance input buffers and open collector drivers. This interface is therefore made up of a combination of two octal receivers and four quad open-collector drivers, all of which are controlled by a PAL.

In order to enforce the inter-access time, the Q-Bus's Reply control line is delayed via a shift register. This delay allows the WD2840 to update the transmission specific event counters in RAM as well as its internal registers.

(f) Reset Circuitry

The WD2840 may be reset by two sources. The first is the PDP's Q-Bus as a result of the cpu asserting the "Bus initialize" line. The second source is a manual reset on the card itself, which may be used during software debugging. Both initiate a monostable which provides the required length reset pulse to the WD2840 and the rest of the circuit.
(g) **Clock and Network Interface Circuitry**

The clock circuitry and the network interface circuit are the same as those used in the micro-computer NIU, as described in the previous section.
Chapter 5

The Network Interface Software

5.1 Introduction

This chapter continues the discussion on the implementation of the network related layers of the ISO RM. The subject is the software which provides the link between the hardware and the Transport/Session Layer interface, i.e. the information has been brought as far as the internal backplane of the computer by the hardware, it is now the task of the software to make it acceptable to the CP/NET software.

![Diagram of network interface software](image)

Figure 5.1 - Chapter Topic

The micro-computer software was not written specifically for CP/NET's NDOS because the interface may be used by students without the CP/NET programs. Thus a simpler interface was provided which would require only a
small adjustment to be compatible with NDOS. These transformation were done by the Token-bus SNIOS, also described below.

The software described for the PDP is simply test software which would form the basis of the device driver. At the end of the chapter the second part of the guidelines for the PDP software designer are given. These guidelines relate to the creation of the device driver to interface the PDP's executive to the LAN hardware described in the previous chapter.

5.2 Software Requirements

5.2.1 The LLC Layer Requirements

There are two types of service specified for the LLC/Network interface. The first is a connectionless service and the second a connection-oriented service. These two forms of service are used to define two types of LLC layers - those providing only a connectionless service, known as Type I LLCs, and those which provide both services, defined as type II LLCs. (The IEEE 802 states that a LLC layer may not simply provide a connection-oriented service.)

In a connection-oriented service, a virtual link is created between the two end nodes before transmission may begin. Each frame is sent along the same route and is acknowledged at the LLC level. At the end of the transmission session, i.e. when the link is no longer required, the link must be disconnected. A connectionless service, on the other hand, is one in which each frame is sent as a separate entity and has to mark its own route across the network. Here flow control and error recovery are provided by the higher layers.

In the IEEE 802 standards, the MAC Layer only provides a connectionless transmission which may either be point-to-point or multi-point and acknowledged or unacknowledged. The LLC layer is entirely responsible for creating the connectionless or connection-oriented service for the higher layers using this service.
If a connection-oriented service had been supplied by this LAN, the LLC Layer would first have had to set up the link between the two end points, where no intermediate nodes exist. The data packets would then have been sent along the link between the two nodes with the extra overhead bytes. Before an acknowledgement could be sent, the destination node would have had to have data to send or would have had to create a special acknowledgement frame.

Sequencing is not a problem on a LAN as the two most remote nodes will generally be less than the frame transmission time apart, i.e. the destination node would have begun receiving the frame before the source node had completed transmission. There are also no alternative routes which would delay one frame more than another. In this LAN, the WD2840 will implement flow control. It would seem unwise to duplicate this service in the higher levels of software. The act of creating and terminating the logical links would also lead to large data overheads when short interactive traffic is predominant.

The provision of a connectionless service appears more practical as:

1. Sequencing is not a problem on a LAN, the nodes are so close.
2. Flow control is provided by the TAC.
3. Each transmission on the LLC Layer is independent thus removing the need for link establishment and disconnection, and consequently reducing overhead.
4. A MAC Layer frame acknowledgement, provided by the TAC, means that this service can ensure frame arrivals.

Connectionless transmission was therefore chosen for this LAN, classifying each LLC entity as a class I LLC. The services provided are to transmit a frame without waiting for an acknowledgement and, as an improvement, to transmit frames with an acknowledgement request.
5.2.2 Parameter Initialization

Before the TAC is used on the network a number of parameters must be initialized. IEEE defines them as:

1. The station's address

2. The address length, implicit in the address value (16 or 48 bits long).

3. The frame acknowledgement time.

4. Token hold time, which defines the maximum time that the TAC is allowed for transmission of frames before the token must be passed.

5. Ring maintenance rotation timer, defining the maximum time allowed for token to circumnavigate the logical ring.

6. The maximum time between solicits for new stations.

7. "In-ring desired", which is a flag set by the host indicating that TAC must attempt to enter the logical ring.

8. Maximum retry limit if the acknowledged connectionless service is used.

There are some deviations from the IEEE standard due the limitations of the WD2840: The address length is only 8 bits instead of 16 or 48 bits long. The maximum retry count is fixed to two by the TAC.

5.2.3 The TAC Buffers

Once the TAC is part of the logical ring on the network the software is responsible for the maintenance of the TAC's buffers. The software must
also be able to flag the higher layers when a data unit arrives, and similarly it should be able to initiate transmission when requested to by the higher layers.

The WD2840 uses a chained buffer structure. On start-up the receive buffer pointer in the Control Block, a section of memory set aside for use by the TAC’s buffer control, is set to point at the start of the first receive buffer. All but the last receive buffer have their pointers, the first two bytes of the buffer, pointing at the address of the next buffer in the chain. The last buffer’s pointer contains zero as a flag. When a data frame is received, the TAC places it into the next available buffer. The DONE bit in the Frame Status Byte (FSB), a byte in the buffer’s control field, is then set and the next buffer pointer in the Control Block is updated. The host cpu is then responsible for reading the buffer, clearing its FSB and linking it to the end of the chain for re-use, by adjusting the previous buffer’s pointer.

![Figure 5.2 - The WD2840 Buffer Structure](image)

The transmitter buffers are dealt with similarly. On start-up the Transmit buffer pointer in the control block is set to zero. When the first buffer is filled by the host cpu, this pointer is set to point at the
first buffer. As each subsequent buffer is filled, the pointer in the top two bytes of the previous buffer are changed from zero to the starting address of the new buffer by the software. Each frame contains a Frame Control Byte (FCB) which enables the control software to specify, amongst other things, whether an acknowledgement is required for that frame or whether this is the last frame. Once the WD2840 has transmitted, or attempted to transmit a frame, it writes one of the status values (given in Table 5.1) to the FSB; after the control software has read the status, the pointers are cleared and the buffer is made available for re-use:

<table>
<thead>
<tr>
<th>FSB Value (in Hexadecimal)</th>
<th>Explanation of the Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Insufficient receiver buffers</td>
</tr>
<tr>
<td>2</td>
<td>Receiver not enabled</td>
</tr>
<tr>
<td>3</td>
<td>Receiver over-run</td>
</tr>
<tr>
<td>4</td>
<td>Frame exceeded 16 receiver buffers</td>
</tr>
<tr>
<td>9</td>
<td>Transmission failure after retry</td>
</tr>
<tr>
<td>A</td>
<td>Transmitter under-run</td>
</tr>
<tr>
<td>B</td>
<td>Too little data supplied by host</td>
</tr>
<tr>
<td>C</td>
<td>Frame exceeded 16 transmit buffers</td>
</tr>
</tbody>
</table>

Table 5.1 - Returned Status Codes

A limitation of the WD2840 is that the Transmit and Receive clocks must not be more than 43% of the clock input to the TAC (which has a maximum of 2.05 MHz) if a single MAC frame is to consist of more than one buffer. By using 320 byte buffers, the largest frame required by the network software package, CP/NET, could be placed into a single buffer. To prevent failure in the event of a larger Network frame being passed to the LLC Layer for transmission, e.g., when CP/NET is not in use, the software segments the frames into 311 byte blocks. (The buffers are 320 bytes long, which includes 8 overhead bytes. Should the total number of data bytes
plus overhead bytes equal 320, the WD2840 will expect a second buffer even though it would contain no data.)

5.2a The Encapsulation of the Messages

Before launching into a description of the software it would be useful to show how the message is encapsulated on the different layers. Each layer was not implemented as a separate entity, e.g. the NDOS performs the Application, Presentation and Session layer functions, thus the interfaces are:

(a) The Application Process/Application Layer interface: At this interface a function is requested, e.g. login or read next sector. The user/application process will supply the function code and variables, in the case of "login" he will give the server's address and the password. The NDOS builds up a message which includes the source and destination addresses, the FNC and the SIZ fields.

(b) The Session/Transport Layer interface: The interface data unit at this interface is as follows:

FMT
DID
SID
FNC
SIZ
Data Byte 0
:
:
:
Data Byte n
The FNC and Data Bytes are the encapsulated data from the application process.

(c) The LLC/MAC Layer interface: The message is then converted into the form required by the hardware control program. At this interface the message looks like this:

Length High
Length Low
Destination Address (DA)
Data bytes

The DA is a copy of the DID provided above, and the data bytes contain the whole message passed across the Session/Transport Layer interface.

(d) The physical network interface: The WD2840 uses the standard HDLC format. This means that the message has been inserted into a frame to facilitate error recovery. The frame is as follows:

Flag (7EH)
Token Control (TC)
Destination Address (DA)
Source Address (SA)
Data bytes
Cyclic Redundancy Check byte
Flag
Network Interface Software
- A Description of the SABus NIU Software

5.3 A Description of the SABus NIU Software

The SABus NIU software was written in 8085 assembler code and consisted of one program - the TAC User Routines (TACUR). This program was divided into five sub-sections, namely, network initialization, message transmission, message reception, status reporting and station isolation.

To provide easy access to the user, the routines may be run by calling one of five addresses at the beginning of the TACUR program. The entry point to the initialization routine would be at the start of TACUR, the entry point for message transmission at an offset of three from the start of the program, and the other routines at subsequent three byte intervals. (The program listing is given in appendix E of this report.)

5.3.1 The Initialization Function

The INITialization routine starts by setting up the parameters required by the TAC. The TAC is then instructed to enter the logical ring. If there is network activity, the NIU checks for a duplicate address (a function provided on the WD2840). If there is no address duplication, the TAC attempts to enter the logical ring at the next opportunity by responding to a token passed to it which will be during a solicit successor phase. The maximum wait is 1 second - the inter-solicit time.

If there is no activity on the network and the TAC has the right to claim the token, it attempts to establish the logical ring by continually polling the whole address space until another node responds. The TAC then informs the user, via the console, that it is part of the network or that it has the same address as another active node, and returns to the calling program.
5.3.2 The Transmission of Frames

The second function, the transmission of frames across the network, is performed by the SNDMSG routine. Two flags are used to indicate the required service. The first is the accumulator which requests transmission with or without waiting for acknowledgements. The second is the 8085’s B-register which indicates whether the data buffer contains a command or a non-command data unit. If B is set to 0, the frame in the data buffer is an Unnumbered Information (UI) frame, in which case the routine adds the IEEE UI Control byte. If the value in the B register is 255 then the data buffer contains a control frame and the SNDMSG routine will not add the UI Control byte as the frame contains its own control byte.

The data is transferred to one of the two WD2840 transmit buffers. Once the buffer header has been set up, the buffer is linked to the previous TAC transmit buffer and transmission is enabled. This routine then waits until the WD2840 signals that it has completed the transmission. The value in the FSB is then read to establish whether or not the transmission was successful. If an error occurred, one of the status codes (given in Table 5.1) is returned in the accumulator, as well as the start address of the last block of data in which the erroneous transmission occurred, contained in the DE register pair. This is to facilitate recovery by the higher levels.

If no error occurred and if there is more data to send, the SNDMSG routine transmits the next 311 byte block until either an error occurs, or the input buffer is empty and then it returns.

5.3.3 The Reception of Data Frames

Because the micro-computers are single user machines, the reception of data from the WD2840 is controlled by polling rather than via interrupts. On entry to the receive message routine (RECMGS), the program first establishes if a data frame has been received. If not, it delays 10 msec, to enable the TAC to update its registers, and then returns with the
accumulator set to 0. If there is data, the RECMG routine reads the frame from the TAC receive buffer into memory.

The IEEE control word is then compared to one of the supported functions, namely Exchange Identification (XID), TEST and Unnumbered Information (UI). If the frame is an XID, the RECMG sends an XID frame which is stored in memory. If the frame is a TEST frame, the received data is looped back to the transmitter. In both the above cases, the Final bit (a bit in the IEEE control byte) is set to the same value as the Poll bit in the received control byte. If the illegal condition of a UI frame with the Poll bit set is received, the frame is read in from the buffer and discarded. In each of the above cases the accumulator is set to zero to indicate that no data was received on procedure return. Finally, if the received frame is a UI frame with the Poll bit cleared (i.e. a valid data frame), the data is read from the TAC input buffer to the output buffer pointed to by the DE register pair. The value of the accumulator is then set to 255 indicating that data was read and the procedure returns.

5.3.4 The Status Report Function

The fourth function provided by the TACUR program is a status report performed by the STAT routine. This routine reads the values of the 16 TAC status/control registers as well as the 11 event counters (which give a count of the occurrence of a number of non-fatal network related events). Each of the counters is set to 0 as it is read. It then returns these values in a buffer pointed to by the DE register pair. This status read enables the application program to calculate the best values for the WD2840 parameters and adjust them while the TAC is on-line.

5.3.5 The Isolate Function

Finally, a function which enables the removal of the TAC from the logical transmission ring is provided. This routine instructs the WD2840 to leave the network. A message is printed to the console and the accumulator
returns a flag to indicate whether the isolation is complete or whether confirmation from the TAC is still pending.

5.4 The Token-bus SNIOS

This program performs the transformation between the standard interface provided by the TACUR program and the interface as required by the CP/NET NDOS program. Its listing may be found in Appendix F of this report.

The TAC User Routines program, described above, provides the user with a LLC/Network layer interface. The SNIOS simply has to transform this interface into one that is acceptable to the NDOS. The overall structure of the SNIOS program is given in the CP/NET Users Guide [14].

The first function of the SNIOS is to initialize the network which is done by calling the TACUR initialize routine. To send a message, SNIOS calls the send message routine in TACUR after adjusting the packet format to conform to the TACUR format. Similarly the reception of a packet is achieved by calling the receive message routine in TACUR and adjusting the format of the packet received.

5.5 The Requirements of the PDP NIU Software

There are two levels of software that make up the device driver. The first level (the only level dealt with in this project) controls the transmission and reception of data frames and the NIU. The second level is responsible for accepting instructions from the RSX-11M Executive and acting on them to provide multi-user access to the network. This level of the software is left for the writer of the PDP 11/23 Network Server Software.

The software used to drive the PDP LAN card is written in LSI-11 assembler code. Because the PDP is used by other students and staff, a
FALCON single user computer, which uses a similar processor and also has the Q-Bus as its internal bus, was used for hardware and software development. Once the higher layers of software are written, the interface card may be plugged directly into the PDP 11/23.

The first requirement of the program is to initialize the hardware according to the IEEE standard as detailed above. The software must then control the TAC, provide correctly formatted buffers, instruct the TAC to send the frames and read frames from the RAM. In order to test the network, some form of user interface had also to be provided. Finally, some code had to be written to service both the Line Time Clock (a real time clock) and WD2840 interrupts.

5.6 A Description of the PDP NIU Software

The program listing may be found in appendix H of this report. It may be broken into 5 sub-sections: control and user interface routines, the device initialization routine, the frame transmission routine, the reception routine and the interrupt service routines.

5.6.1 User Interface and Control Routine (MAIN)

The MAIN routine, which may be thought of as the station management, has overall control of the program. It begins by calling the initialization routine to set up the parameters required by the TAC. The initialization routine instructs the TAC to go on-line and attempt to enter the logical loop. When it returns to the MAIN routine, the user is informed of the status and is prompted to select either data transmission or reception.

While the computer is waiting for a response from the user, it continually checks to see if a data frame interrupt has been received. If an interrupt has occurred, or if the user chooses to receive a frame from the network, the MAIN routine calls the message receive (RECMSC) routine.
Once the TAC has completed reception, RECMGS reads the TAC's buffer into a temporary input buffer which is used to display the message on the console.

If the user chooses to send a frame, he is prompted for the destination address and the message, SNIDMSG, is then called. This routine sets up the buffers, links them to the previous ones and enables TAC transmission. Once transmission of the frame is complete, it returns a status report to MAIN procedure.

5.6.2 Initialization Routine (INITNET)

This routine has the task of including the TAC in the logical ring on the network. It first sets up the WD2840 registers and then checks for network activity by sensing the LAN for a fixed interval defined by the Line Time Clock. If no Network Dead interrupt occurred during this period, the network is presumed to be active. If this is not the case, the TAC is instructed to attempt to start up the network by initiating the scanning procedure. The TAC continues scanning until it receives a response from another TAC.

On an active network, a check is made for a duplicated station address. This is done by setting a timer to the solicit successor interval. If only one token interrupt occurs in this period, it is assumed to be as the result of a solicit. Should more than one interrupt occur, an active station must have the same address as the station attempting to enter the logical ring on the network. More than one interrupt would occur in the duplicated address situation as the active station is addressed each token rotation, which must be within the token rotation time, which, in turn, is less than the inter-solicit interval. In the duplicate address case the TAC is disabled and the station management informed.

Once it has been established that there is no address duplication, the TAC attempts to enter the logical ring on the network, i.e. the TAC will respond to the next token addressed to it. Once the logical loop is established, INITNET returns to the MAIN routine.
5.6.3 Transmission Control Routine (SNDMSG)

The SNDMSG routine controls the transmission of frames. It first prompts the user for both the destination address and the message, which is limited by the software to 650 bytes. Once the message has been read, the FILTXBUF routine, which transfers the data from an internal buffer to one of the TAC's buffers in dual-port RAM, is called to create a frame and send it to the TAC transmit buffers (located in the dual-port RAM) in the required format. The FILTXBUF routine then instructs the TAC to send the frame and waits for its completion. This routine then returns the transmission status in the RI register. The returned status values are the same as those used in the SABus software, as listed in Table 5.1.

The FILTXBUF routine is written so that it can be used in the device driver and so makes no assumptions about the size of the transmission buffer. It segments the buffer received into frames of 311 bytes long and adds the IEEE control byte specified in the CONTTYPE variable to each frame.

The SNDMSG routine waits for FILTXBUF to return and then returns to the MAIN routine.

5.6.4 Receive Message Routine (RECMSG)

This routine checks for a full receive buffer. If none is found it returns. Should a buffer be available, it clears both the FSB and the appropriate bit in the interrupt register and then calls the Read TAC Receive Buffer (READWDBUF) routine in order to transfer a frame from one of the TAC's buffers to an internal buffer for screen display.

The READWDBUF routine finds out which buffer should have been written to and checks the FSB for confirmation. If the DONE bit in the FSB is not set, either this buffer contains an aborted frame, which means that the next buffer may be full, or no data was received. If neither this buffer or the next has its DONE bit set, the READWDBUF routine returns with the
NODATA flag set to true. If the DONE bit is set, the READWDBUF routine reads the data to the input buffer (INBUF) and links the TAC buffers in dual-port RAM to the chain for re-use by the WD2840.

In order to ensure that all the buffers are read, the READWDBUF routine reads the FSB of the following two frames to see if the data contained in them is ready for access. If this is the case then the READWDBUF sets the NOMOREDATA flag to FALSE.

If the NODATA flag is FALSE the RECMSG routine writes the frame source address, from the first byte of INBUF, to the screen after converting it to two ASCII characters. The rest of the message in INBUF is then sent directly to the console. NOMOREDATA is checked to ascertain if READWDBUF must be called to read a second buffer. Once all the available frames have been read, RECMSG returns to the MAIN routine.

5.6.5 Interrupt Service Routines (LTCSR and WDSR)

These two service routines deal with interrupts from the two interrupt sources, the Line Time Clock and the NIU card.

(a) LTCSR

The Line Time Clock Service Routine (LTCSR) provides a means of calculating long delays, e.g. during duplicate address checking, and ensures, every second, that new successors are solicited.

To create a means of calculating long delays, the 20 msec hardware timer interrupt is enabled. The interrupt service routine simply increments a counter each time an interrupt occurs. The counter is then read by other sections of the software when long delays are required. After one second, i.e. 50 interrupts, the interrupt service routine initiates a scan procedure by setting the TAC's successor address to its own plus one.
(b) WDSR

Because the TAC is receiving frames from the network at 1 Mbps, multiple interrupts to the same interrupt routine may occur. On the PDP this would not cause failures as the interrupt service routine may increase its priority, thereby excluding further interrupts to the same location until service is complete. The FALCON computer does not permit interrupt priorities and so multiple interrupts become problematic. To overcome this, the interrupt handler was made as short as possible.

The Western Digital Service Routine (WDSR), which services any interrupts from the NIU card, is called when the WD2840 initiates an interrupt. The WDSR reads the value of the TAC's interrupt register (IRO) and sets the appropriate bit, or bits, in the IRDATA register in the PDP's RAM. The IRDATA register is read by the other subroutines to establish if a particular interrupt has occurred.

The IRO register must be read twice in order to clear the interrupt because of faults in the prototype version of the WD2840.

5.7 Guidelines for the PDP Server Program (Part 2)

The PDP device driver forms the link between the interface hardware and the PDP's Operating System Executive (that section of the operating system which directs program execution). The purpose of the device driver is to control the interface hardware thereby allowing the operating system to perform a number of standard functions, e.g. reading and writing of blocks of data.

The device driver may be broken into 5 sub-sections, each requiring a separate entry point for the executive. These sub-sections are:

1. I/O Initiator
2. Device interrupt
3. Device timeout
4. Cancel I/O
5. Power failure

The I/O initiator is called to "wake-up" the device driver when a task queues one of the functions. For instance, to send a frame, a task would queue a write block request to the executive. The executive would then call the device driver at the I/O initiator entry point giving details of the function to be performed.

Once the device has completed the function it would interrupt the cpu which would call the Device Interrupt entry point. This section of the driver would thus have code to deal with device interrupts.

The Device Timeout entry point is called to cancel an I/O function which has been active too long.

The Cancel I/O entry point provides a point for the executive to call in order to cancel a previously initiated I/O function call. This is generally as a result of a task command rather than an executive timeout.

When power is restored and a device driver was running before failure, the Power Failure entry point may be called to initialize the device hardware. The executive may also be instructed to call this entry point when the driver is loaded.

Sections of the PDP's NIU software, described above, may be incorporated into the device driver. Firstly, the I/O initiator would have to be able to perform a number of functions, e.g. transmission and reception of blocks of data (or frames), device initialization and reading and adjusting certain parameter values. (The allowed functions must be specified within the driver software.) Three of the NIU software routines may be called by the initiator:

1. INITNET - to start-up the network.
2. SNDMSG - to initiate the transmission of frames.
3. **RECMG** - to read a frame received from the network.

Secondly, the power failure routine could call **INITNET** to restart the interface hardware.
Chapter 6

Network Verification and Conclusions

This chapter examines the results gathered from tests performed on the network from which a number of conclusions are drawn.

6.1 Network Verification

The LAN was constructed as described in the previous chapters. Circuits were laid out on printed circuit board, constructed and tested. A test rig was then set up, as shown in Figure 6.1, to verify network functioning.

![Figure 6.1 - The Token-bus LAN Test Rig](image)

6.1.1 Functional Performance

Because the PDP Server program has not yet been written, three tests were necessary to confirm the correct functioning of the partially complete LAN.
Network Verification and Conclusions

Firstly, the implementation of CP/M and the CP/NET programs on the U.C.T. micro-computers was verified. This was done by connecting two of the computers via separate RS-232 links to an MP/M-II server. Using Digital Research software to control the network oriented layers, and thus provide serial communication, the U.C.T. micro-computers were able to successfully communicate with the MP/M Server and use the utilities provided. This verified CP/M because CP/NET loaded and ran correctly. All the CP/NET functions were used. They worked correctly, thus confirming correct implementation of the CP/M and NDOS part of CP/NET on the U.C.T. micro-computers.

Secondly, the implementation of IEEE layers was tested using a specially constructed network, as illustrated in Figure 6.1. Using the software described in chapter 5, messages of varied lengths and contents were successfully transmitted between the micro-computers and the Falcon. The inter-solicit timers were tested by instructing one of the computers off the network and then immediately back on again. The computer was included in the logical loop within the inter-solicit time of 1 second. It was discovered that the duplicate address checking facility provided on the WD2840 prototype versions did not function correctly. A duplicate station address thus led to the exclusion of one of the stations with a duplicated address. This portion of the program was not adjusted as it would function correctly with future versions of the TAC.

Finally, the network as a whole and its interaction with CP/NET was tested by allowing the micro-computers to issue CP/NET requests to the Falcon, which was acting as the Server. Manual Server responses were provided from the Falcon's keyboard. In all cases the network was found to function correctly.

6.1.2 User Level Software

Briefly the user level requirements for the training network were to provide:
Network Verification and Conclusions

1. Program capture
2. Assembly and debugging of assembler code
3. Compilation of high-level languages programs
4. File storage and recovery
5. Electronic Mailing and Word processing facilities

The above requirements were implemented by the combination of CP/NET and CP/M on the micro-computers and will be provided by the software on the PDP.

To facilitate program capture a CP/M editor, e.g. MicroSoft's WordMaster, could be used. This program was run over the RS-232 test network and performed correctly. Other CP/M programs could provide the assembler and debugging facilities, e.g. ASM for the assembler facility and SID or DDT as a debugger. Any of these programs could run under CP/M on the micro-computers. There were also a number of high-level languages supported by CP/M, e.g. PASCAL MTL, FOURTH, PL/M and many others, which could be used on the network.

The assembler debugging program, SID, was tested during the token-network development phase. This program was run from the EPROM disk. The Electronic Mailing facility is part of CP/NET and was also tested with the help of the RS-232 test network. File storage and recovery were tested in the same way. A word processor, WordStar, was also run over the RS-232 network.

6.1.3 Physical Performance

The physical requirements for the network were:

1. It should span 800 m.
2. It had to support up to 30 terminals.
3. A speed requirement of 1 Mbps.
Figure 6.2 shows the data signalling rate versus cable length for a CCITT V.11 transmission (an equivalent of the EIA RS-422 standard). It is clear that even for a terminated cable (curve 1), 800m is too long. To achieve this distance, the 800m length was broken into 75 m sections interconnected by bi-directional RS-422 repeaters.

![Diagram showing data signalling rate versus cable length for balanced transmission.](image)

Figure 6.2 - Data Signalling Rate versus Cable Length for Balanced Transmission

To ensure correct transmission over 75 m, a distance test was performed by connecting a 25 m section of 3 pair cable between the two micro-computers in the test rig. The cable was connected so that the signal travelled 75 m by joining two pairs at each end. The signal travelled the required 75 m in a noisy environment, inducing noise on itself, without an increase in the frame retransmissions due to noise.
The second requirement could not be tested as there were not 30 terminals or line drivers available. The specification for the RS-422 driver used states that 32 drivers can be used on a common link. The software addressing makes provision for 32 nodes on the bus and so, although this feature was not tested, if the chip performs according to specification, the requirement will have been fulfilled if the chip is found to perform according to specification.

6.1.4 Data Traffic Performance

Because the PDP server software is not complete, it is difficult to verify the total network performance. The LAN uses a 1 Mbps transmission rate and so the file transfer rate will be far higher than that of the original network. As far as queuing delays are concerned, it can only be said, at this stage that the file transfer time for a 44k byte frame is about 2 seconds on the test rig, with two micro-computers transmitting simultaneously. The PDP software author will have to provide an access fast enough to keep the delay to a minimum.

6.1.5 Administration Parameters

The TACUR program provides a status report which supplies the following parameters:

1. The number of scan frames received.
Network Verification and Conclusions
- Network Verification

2. The number of transmission failures on first attempt which were successful on the second attempt.

3. The number of erroneous access control frames received.

4. The total number of Negative Acknowledgements sent.

5. A count of the invalid frames received due to abnormal network conditions.

6. The number of duplicate tokens received.

7. A duplicate address count.

8. The number of aborted transmissions.

9. The number of times the network has been inactive for the timeout period.

These values, in conjunction with a real time clock and a software counter to count the number of frames sent and received, can be used to supply a variety of data for monitoring and control purposes. The software will form part of the PDP server software.

6.1.6 Future Requirements

The network should be able to expand to accommodate a second mini-computer, voice and video data and the connection of other terminal equipment.

The second mini-computer could be included by using a version of the network interface hardware and the driver software written for the PDP.

Should the baseband transmission be replaced by the FSK broadband transmission, and the voice and video interface circuits are constructed, this network could certainly support multiple data channels over a coaxial cable bus.

Any data terminating equipment may be connected to the network. To achieve this a version of the network interface unit would have to be constructed, and the DTE would have to be "intelligent" enough to drive it.
Network Verification and Conclusions
- Compliance with the IEEE 802 Standards

6.2 Compliance with the IEEE 802 Standard

The only section of the project that complies with the IEEE 802 standards is the LLC layer, which implements a Class I LLC layer.

The MAC layer does not comply to the IEEE 802.4 standard because of the use of the WD2840 TAC. The TAC was used because of its low cost, availability and ease of use.

On the PHY layer, the network was originally designed to comply with a IEEE 802.4 specification for Phase-continuous FSK. This design was based on the use of a pair of FSK modem chips which were not produced in time for use in this project. The PHY layer was thus made to consist of a bus of twisted pair wire which is driven by RS-422 drivers, which does not comply with the IEEE 802.4 standard.

6.3 Conclusions

The introductory chapter high-lighted three reasons for the development of a LAN in the Department of Electrical and Electronic Engineering at U.C.T., these were:

1. The old network was slow.
2. It was restricted in its switching ability.
3. High level software was too expensive, and, if used, would overload the PDP even more.

In principle, it is best to adhere precisely to an international standard, but the design for each LAN is determined by the requirements for that network. To adhere precisely to the IEEE 802 standards in this case would have been too costly, time consuming and complex. By following the principles of the 802 standards and by using the WD2840 network controller, the beginnings of an inexpensive yet satisfactory LAN were created.
The 1 Mbps Token-passing bus successfully overcomes the three drawbacks of the original network and makes provision for the supply of a micro-computer training facility for a number of years.
References


References


Additional Reading List

Below is a list of references that provided a background for the development of this LAN:

On the subject of other similar LANs:


On the subject of LAN performance:


APPENDIX A

The BIOS Program for the UCT Micro-computers
**CP/M BIOS FOR UCT SABUS KITS**

**Program Language**: 8085 Assembler Code

**Author**: Adapted from Microcom BIOS by Q.P. McNerney

**Date**: 9 May 1985

---

**KBIOS is the modified version of the CP/M BIOS to enable the use of CP/M in the UCT built SABus Kits.**

The KBIOS may be divided into the following sections:

1. **The jump table**
2. **The disk definitions**
3. **The boot routines**
4. **The disk package**
5. **The communications package**
6. **The communications variables**
7. **The disk variables**

---

1. The jump table is the entry point for the system calls to the program.
2. The disk definitions are to enable the BIOS to access the disk, as it needs to know the format of the data.
3. The boot routines are there to enable the system start up.
4. The disk package is the section that deals directly with the reading of the data from the mass storage card.
5. The communications package, this section provides a tool for communicating with any type of terminal and printer. The package enables the use of the console port, USART 1 and the parallel port, in a variety of formats.

6. Communication variables,

7. Disk variables, for the storage of variables.
BIAS is address offset from 3400H for a memory system other than 16k.

0036 = MSIZE EQU 54 ; 54k system
8800 = BIAS EQU (MSIZE-20)*1024 ; The extra bias required
B500 = CCP EQU 0DD00H - (64-MSIZE)*1024 ; CCP base address
BD06 = BDOS EQU CCP+806H ; BDOS base address
CB00 = CBIOS EQU CCP+1600H ; CBIOS base address
0004 = CDISK EQU 0004H ; Current disk storage space
0003 = IOBYTE EQU 0003H ; The IO Byte address
SECTION 1 - BIOS JUMP TABLE

Jump vectors for CBIOS subroutines

<table>
<thead>
<tr>
<th>Vector</th>
<th>Address</th>
<th>Subroutine</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>C30301</td>
<td>JMP BOOT</td>
<td>Cold boot entry</td>
</tr>
<tr>
<td>0003</td>
<td>C37F01</td>
<td>WBENTRY JMP WBOOT</td>
<td>Warm boot entry</td>
</tr>
<tr>
<td>0006</td>
<td>C3E601</td>
<td>JMP CONST</td>
<td>Console status</td>
</tr>
<tr>
<td>0009</td>
<td>C3F001</td>
<td>JMP CONIN</td>
<td>Console input</td>
</tr>
<tr>
<td>000C</td>
<td>C3FA01</td>
<td>JMP CONOUT</td>
<td>Console output</td>
</tr>
<tr>
<td>000F</td>
<td>C34002</td>
<td>JMP LIST</td>
<td>List output</td>
</tr>
<tr>
<td>0012</td>
<td>C31002</td>
<td>JMP PUNCH</td>
<td>Punch output</td>
</tr>
<tr>
<td>0015</td>
<td>C31C02</td>
<td>JMP READER</td>
<td>Reader input</td>
</tr>
<tr>
<td>0018</td>
<td>C38401</td>
<td>JMP HOME</td>
<td>Home drive</td>
</tr>
<tr>
<td>001B</td>
<td>C39001</td>
<td>JMP SETDSK</td>
<td>Set disk</td>
</tr>
<tr>
<td>001E</td>
<td>C3A101</td>
<td>JMP SETTRK</td>
<td>Set track</td>
</tr>
<tr>
<td>0021</td>
<td>C3AD01</td>
<td>JMP SETSEC</td>
<td>Set sector</td>
</tr>
<tr>
<td>0024</td>
<td>C3BB01</td>
<td>JMP SETDMA</td>
<td>Set DMA</td>
</tr>
<tr>
<td>0027</td>
<td>C3C301</td>
<td>JMP READ</td>
<td>Read the described sector</td>
</tr>
<tr>
<td>002A</td>
<td>C3E001</td>
<td>JMP WRITE</td>
<td>Write the described sector</td>
</tr>
<tr>
<td>002D</td>
<td>C32602</td>
<td>JMP LISTST</td>
<td>List device status</td>
</tr>
<tr>
<td>0030</td>
<td>C3E301</td>
<td>JMP SECTRN</td>
<td>Sector translate</td>
</tr>
</tbody>
</table>
### SECTION 2 - DISK DEFINITIONS

<table>
<thead>
<tr>
<th>Disk header definition</th>
<th>Translation vector address (0=none)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSKRDEF:</td>
<td>DW</td>
</tr>
<tr>
<td>SCR:</td>
<td>DW</td>
</tr>
<tr>
<td>ALV:</td>
<td>DW</td>
</tr>
<tr>
<td>DSKDEF:</td>
<td>DW</td>
</tr>
<tr>
<td>SCR:</td>
<td>DW</td>
</tr>
<tr>
<td>ALV:</td>
<td>DW</td>
</tr>
</tbody>
</table>

**Disk header definition**
- **DSKRDEF**: Disk scratch area
- **SCR**: Check sum scratch area
- **ALV**: Allocation information

**Disk definition block pointer**
- **DB**: Here Sectors/128=Address on the eprom
- **BSH**: Block shift factor
- **BLM**: 2k long~
- **EXM**: The extent mask, determined by the total number of blocks on this drive.
- **DSM**: The maximum number of blocks on this drive.

**Translation vector address**
- **0=none**
- **Translation vector address**

**Scratch area**
- **Scratch area**
- **DSKRDEF**
- **CSCR**
- **ALV**
- **DB**
- **DSKDEF**
- **DB**

**Directory scratch area**
- **DSKRDEF**
- **CSCR**
- **ALV**
- **DB**

**Check sum scratch area**
- **DSKRDEF**
- **CSCR**
- **ALV**
- **DB**

**Allocation information**
- **DSKRDEF**
- **CSCR**
- **ALV**
- **DB**

**64 Sectors per Track**
- **EXM**
- **BSH**
- **BLM**

**Block shift factor**
- **BSH**
- **BLM**
- **EXM**

**Total number of blocks**
- **DSM**
- **BSH**

**Max number of directory entries**
- **DSM**

**Blocks reserved for directory entries**
- **DSM**

**The number of checked directory entries**
- **DSM**

**0x00000000B**

---

#05 UCT BIOS ADAPTED FOR MASS STORAGE CARD ACCESS

---

**ASSEM 1.1**

---

**RMAC**

---

**CP/M**
FUNCTION: BOOT;

Inputs: None
Outputs: None
Destroys: A, B, C, D, E, H, L, Flags
Calls: WBOOT, CONOUT

Description: This routine gets called by the cold start loader which had loaded CPM. IOBYTE is initialised and the signon message is displayed. The blocking parameters are initialised and the entry points of the operating system set. The cold start autoload is placed into the CCP and the CCP is called.

OEPNUM EQU 80H ; The old EPROM number
INTIOBY EQU 10010100B ; LST=LPT, PUN=PTP, RDR=PTR, CON=TTY
ESC EQU 1BH ; The ASCII ESC value
COMLINE EQU CCP+7H ; Command line for auto entry

BOOT:

0103 3A8000 LDA OEPNUM ; Read the old number
0106 3D DCR A ; decrement by one
0107 32B03 STA EPOSET ; and write it to the new position
CP/M RMAC ASSEM 1.1 #007 UCT BIOS ADAPTED FOR MASS STORAGE CARD ACCESS

010A CD8702 CALL BOOTCOM ; Set up the Communications
010D 0621 MVI B,MESLEN ; Send the Sign on message
010F 215E01 LXI H,MESSAGE ;

MESLOOP:
0112 E5 PUSH H ;
0113 C5 PUSH B ;
0114 4E MOV C,M ; Read the Byte and output it
0115 CDFA01 CALL CONOUT ;
0118 C1 POP B ;
0119 E1 POP H ; Next byte
011A 23 INX H ;
011B 05 DCR B ; End?
011C C21201 JNZ MESLOOP ; No - loop
011F 3EC3 MVI A,(JMP) ; Set up the values required for CP/M:
0121 320000 STA 0 ; 1. At 0000H - JMP Warm_Boot_entry
0124 210300 LXI H,WBENTRY ;
0127 220100 SHLD 1 ; 2. At 0005H - JMP BDOS_entry
012A 320500 STA 5 ; Set the current disk to A:
012D 2106BD LXI H,BDOS ;
0130 220600 SHLD 6 ;
0133 AF XRA A ;
0134 320400 STA CDISK ;
0137 4F MOV C,A ;
0138 CD3E01 CALL AUTOLOAD ; Gives the CCP the program to be
013B C300B5 JMP CCP ; loaded on entry

; AUTOLOAD:
013E 2107B5 LXI H,COMLINE ; Start CP/M via CCP
0141 115201 LXI D,PRONAME ; Command line space
0144 010C00 LXI B,LENG ; Name of the program to be entered

AUTOLOOP:
0147 1A LDAX D ; Read the name
0148 77 MOV M,A ; Save it in the correct space
0149 23 INX H ; adjust the pointers
CP/M RMAC ASSEM 1.1  #008  UCT BIOS ADAPTED FOR MASS STORAGE CARD ACCESS

014A 13  INX  D  ; End of the line?
014B 0B  DCX  B  ;
014C 78  MOV  A,B  ;
014D B1  ORA  C  ;
014E C24701  JNZ  AUTOLOOOP  ; No - then loop
0151 C9  RET  ; or return

; PRONAME:
0152 0A413A4350  DB  0AH,'A:CPNETLDR',00H
000C =  LENG  EQU $-PRONAME

015E 1B481B4A  MESSAGE  DB  ESC,'H',ESC,'J' ; Clear the console
0162 2020202020  DB ' UCT - CP/NET System',0DH,0AH
0021 =  MESLEN  EQU $-MESSAGE

;-------------------------------------------------------------------------------
; THE WARM BOOT IS NOW UNDER NDOS CONTROL
;-------------------------------------------------------------------------------

WBOOT:
017F 21FFFF  LXI  H,0FFFFH  ; As an error indicator
0182 7C  MOV  A,H  ;
0183 C9  RET  page
SECTION 4 - DISK PACKAGE

Main Procedures

HOME: Set the constant address values to zero address
SELDSK: Check the local disk selection is A:
SETTRK: Choose EPROM
SETSEC: Set the addresses for the 128 byte pages
SETDMA: Set dma address
READ : Read a block of data from the EPROM to memory
WRITE: Not available
SECTRAN: Has no effect

EPNUM EQU 23H ; Mass storage card: EPROM number latch (MSB)
EPADRHI EQU 22H ; Address byte latch (MSB)
EPADRLO EQU 21H ; Address byte latch (LSB)
EPDATA EQU 20H ; Data register

0023 = 0022 = 0021 = 0020 =
FUNCTION: HOME;

Inputs: None
Outputs: None
Destroys: A,H,L,flags
Calls: None
Description: Reads the Eprom offset and sets the Eprom number to this value.

HOME:

LDA EPOSET ; Read the EPROM offset value
OUT EPNUM ; Send it to the EPROM Number latch
MVI A,0
OUT EPADRHI ; Zero to the address latches
OUT EPADRLO
RET

page
FUNCTION SELDSK

Inputs: C: Selected disk number
Outputs: HL: base of disk parameter block
HL: 0 if an error occurred
Destroys: A,H,L, flags
Description: The disk number is set and, if valid, the base
of the disk parameter block is returned in HL or
if not, HL = 0.

SELDSDK:

0190 79 MOV A,C ; Make a note of the requested disk
0191 328903 STA SEKDSK ;
0194 FE00 CPI 0 ;
0196 C29D01 JNZ ENDSLELD ; If it is not 0 (A:) then error
0199 213300 LXI H,DSKHED ; The base of the Disk Parameter Header
019C C9 RET ENDSLELD:

019D 210000 LXI H,0 ; H=0 means error
01A0 C9 RET ;
FUNCTION: SETTRK

Inputs: BC: Selected track number
Outputs: None
Destroys: A
Calls: None
Description: Set the EPROM number for later use.

SETTRK:

01A1 E5  PUSH  H ; Destroy as little as possible
01A2 3A8B03 LDA  EPOSET ; Read the offset
01A5 81  ADD  C ; The track number, = EPROM number
01A6 328C03 STA  EPNUMS
01A9 D323 OUT  EPNUM ; Set the EPROM number
01AB E1  POP  H
01AC C9  RET

page
FUNCTION: SETSEC

Inputs: BC: Selected sector number
Outputs: None
Destroys: H,L
Calls: None
Description: Sets up the MSB value on the EPROM card and calculates the LSB address offset.

SETSEC:

01AD 79 MOV A,C ; Read the sector number
01AE B7 ORA A ; Clear the carry flag
01AF 1F RAR ; £128
01B0 47 MOV B,A ; B = 0 as max of 64 tracks
01B1 3E00 MVI A,0 ; Get LSB offset
01B3 1F RAR ;
01B4 328A03 STA ADROFF ; Save the offset.
01B7 78 MOV A,B ; Set the MSB of the EPROM
01B8 D322 OUT EPADRHI ;
01BA C9 RET ;

page
FUNCTION: SETDMA

Inputs: BC: Selected DMA address
Outputs: None
Destroy: None
Calls: None
Description: Set the selected DMA address.

SETDMA:

01BB E5     PUSH   H
01BC 60     MOV     H,B
01BD 69     MOV     L,C
01BE 228D03 SHLD    DMAADR
01C1 E1     POP     H
01C2 C9     RET

page
FUNCTION READ

Inputs: None

Outputs: A:=0 => no error
A:=1 => error

Destroys: All

Calls: None

Description: Reads a 'Sector' of data from the EPROM card and puts it into the buffer pointed to by the DMAADR pointer.
A zero in the A register means that there was no error a 1 means an error occurred.

READ:

01C3 2A8D03
01C6 BB
01C7 0E00
01C9 3A8A03
01CC 47
01CD 3E00

RL1:

01CF B0
01D0 D321
01D2 DB20
01D4 12
01D5 13
01D6 0C
01D7 79
01D8 FE80
01DA DACF01
01DD 3E00
01DF C9

LHLD DMAADR ; Reads the address of the buffer
XCHG
MVI C,0 ; Counter
LDA ADROFF
MOV B,A ; Save it in the B register
MVI A,0

RL1:
ORA B ; Include the offset bit
OUT EPADRLO ; Send it to the EPROM card
IN EPDATA ; Read the byte at that address
STAX D ; Store in the memory buffer
INX D
INR C
MOV A,C ; Check for 128
CPI 128
JC RL1
MVI A,0 ; No error
RET
;-------------------------------------
;            FUNCTION WRITE            
; Inputs: 
; Outputs: 
; Destroys:  A 
; Calls:    None 
; Description: This routine should not be called and so there is an error return if it is called.
;-------------------------------------

WRITE:
01E0 3E01
01E2 C9
MVI A,1 ; As an error indicator
RET ;
page
FUNCTION SECTRAN

Inputs:
BC:=sector to translate
HL:=translated sector

Outputs: None

Description: The sector translation is not performed until debllocking and is therefore returned unchanged.

SECTRAN:

MOV H,B
MOV L,C

Simply duplicate the sector
SECTION 5 - Communications Package

Main Procedures

- CONST : Console status
- CONIN : Console character in
- CONOUT : Console character out
- LIST : List character out
- PUNCH : Punch character out
- READER : Reader character in
- LISTST : List status

Auxiliary Procedures

- BOOTCOM : Initialise communications ports and protocols
- SELDEV : Select hardware device
- RDRST : Reader status to transmit
- SRSER0 : Status serial-0 port to receive
- STSERS : Status serial-0 port to transmit
- SRSER1 : Status serial-1 port to receive
- STSER1 : Status serial-1 port to transmit
- STPARL : Status parallel port to transmit
- INSER0 : Input serial-0 port
- INSER1 : Input serial-1 port
- OUTSER0 : Output serial-0 port
- OUTSER1 : Output serial-1 port
### UCT BIOS ADAPTED FOR MASS STORAGE CARD ACCESS

<table>
<thead>
<tr>
<th>Address</th>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0000$</td>
<td>COMCARD EQU</td>
<td>Base address for the Communications Port</td>
</tr>
<tr>
<td>$0001$</td>
<td>SER0$D EQU</td>
<td>Console port - Data register</td>
</tr>
<tr>
<td>$0002$</td>
<td>RXRDY EQU</td>
<td>Receive Ready bit mask</td>
</tr>
<tr>
<td>$0003$</td>
<td>IOBYTE EQU</td>
<td>The IO Byte address</td>
</tr>
<tr>
<td>$0004$</td>
<td>SER0$C EQU</td>
<td>Console port - Command register</td>
</tr>
<tr>
<td>$0005$</td>
<td>SER1$D EQU</td>
<td>Serial Port 1 - Data register</td>
</tr>
<tr>
<td>$0006$</td>
<td>PARL$A EQU</td>
<td>Parallel Port - Port A</td>
</tr>
<tr>
<td>$0007$</td>
<td>SER1$C EQU</td>
<td>Serial Port 1 - Command register</td>
</tr>
<tr>
<td>$0008$</td>
<td>PARL$B EQU</td>
<td>Parallel Port - Port B</td>
</tr>
<tr>
<td>$0009$</td>
<td>SER1$M EQU</td>
<td>Serial Port 1 - Status register</td>
</tr>
<tr>
<td>$000A$</td>
<td>PARL$C EQU</td>
<td>Parallel Port - Port C</td>
</tr>
<tr>
<td>$000B$</td>
<td>COMCARD+0</td>
<td>Serial-1 ETX/ACK protocol transmit</td>
</tr>
<tr>
<td>$000C$</td>
<td>COMCARD+1</td>
<td>Serial-1 X-ON/X-OFF protocol transmit</td>
</tr>
<tr>
<td>$000D$</td>
<td>COMCARD+4</td>
<td>Serial-1 ETX/ACK protocol status</td>
</tr>
<tr>
<td>$000E$</td>
<td>COMCARD+5</td>
<td>Serial-1 X-ON/X-OFF protocol status</td>
</tr>
<tr>
<td>$000F$</td>
<td>COMCARD+12</td>
<td>Serial-1 ETX/ACK protocol status</td>
</tr>
<tr>
<td>$0010$</td>
<td>COMCARD+13</td>
<td>Serial-1 X-ON/X-OFF protocol status</td>
</tr>
<tr>
<td>$0011$</td>
<td>COMCARD+15</td>
<td>Serial-1 ETX/ACK protocol status</td>
</tr>
<tr>
<td>$0012$</td>
<td>COMCARD+16</td>
<td>Serial-1 X-ON/X-OFF protocol status</td>
</tr>
<tr>
<td>$0013$</td>
<td>COMCARD+17</td>
<td>Serial-1 ETX/ACK protocol status</td>
</tr>
<tr>
<td>$0014$</td>
<td>COMCARD+18</td>
<td>Serial-1 X-ON/X-OFF protocol status</td>
</tr>
<tr>
<td>$0015$</td>
<td>COMCARD+19</td>
<td>Serial-1 ETX/ACK protocol status</td>
</tr>
<tr>
<td>$0016$</td>
<td>COMCARD+20</td>
<td>Serial-1 X-ON/X-OFF protocol status</td>
</tr>
<tr>
<td>$0017$</td>
<td>COMCARD+21</td>
<td>Serial-1 ETX/ACK protocol status</td>
</tr>
<tr>
<td>$0018$</td>
<td>COMCARD+22</td>
<td>Serial-1 X-ON/X-OFF protocol status</td>
</tr>
<tr>
<td>$0019$</td>
<td>COMCARD+23</td>
<td>Serial-1 ETX/ACK protocol status</td>
</tr>
<tr>
<td>$001A$</td>
<td>COMCARD+24</td>
<td>Serial-1 X-ON/X-OFF protocol status</td>
</tr>
<tr>
<td>$001B$</td>
<td>COMCARD+25</td>
<td>Serial-1 ETX/ACK protocol status</td>
</tr>
<tr>
<td>$001C$</td>
<td>COMCARD+26</td>
<td>Serial-1 X-ON/X-OFF protocol status</td>
</tr>
<tr>
<td>$001D$</td>
<td>COMCARD+27</td>
<td>Serial-1 ETX/ACK protocol status</td>
</tr>
<tr>
<td>$001E$</td>
<td>COMCARD+28</td>
<td>Serial-1 X-ON/X-OFF protocol status</td>
</tr>
<tr>
<td>$001F$</td>
<td>COMCARD+29</td>
<td>Serial-1 ETX/ACK protocol status</td>
</tr>
</tbody>
</table>

**Note:** The symbols indicate the base address for various ports and registers, with specific addresses for different protocols and status conditions.
FUNCTION CONST

Inputs: None

Outputs: A:=00 => ready
A:=FF => not ready

Destroys: A,D,E,F,H,L

Calls: SELDEV

Description: Return status of console device in "A" register.

CONST:

01E6 213202 LXI H,#CONST ; Base address of the Jump table
01E9 3A0300 LDA IOBYTE ; Select the device
01EC 07 RLC ; $2
01ED C37202 JMP SELDEV ; Jump to the jump table calculation
FUNCTION CONIN

Inputs: None

Outputs: A: character from console

Destroys: A,D,E,F,H,L

Calls: SELDEV

Description: Waits until a character is ready and then returns it in the "A" register.

CONIN:

$01F0 213A02  LXI  H,@CONIN  ; Base address for the consoles
$01F3 3A0300  LDA  IOBYTE  ; Adjust using the IO BYTE
$01F6 07  RLC  $01F7 C37202  JMP  SELDEV  ; and jump

page
FUNCTION: CONOUT;

Inputs: C: character to send

Destroys: A,D,E,F,H,L

Calls: SELDEV

Description: Output a character to the console device.

----------

CONOUT:

LXI H,@CONOUT ; Console output base address
LDA IOBYTE ; Adjust the address using the IO Byte
RLC ; $2
JMP SELDEV ;

page
FUNCTION: LIST

Inputs:  C: character to list
Outputs: None
Destroys: A,D,E,F,H,L
Calls:   SELDEV
Description: Output a character to the listing device.

LIST:

0204 214A02  LXI  H,@LIST ; Base address for the List devices
0207 3A0300  LDA  IOBYTE ; Adjust using the IO Byte
020A 07     RLC
020B 07     RLC
020C 07     RLC
020D C37202 JMP  SELDEV ;
FUNCTION: PUNCH

Inputs: C: character to punch
Outputs: None
Destroys: A, D, E, F, H, L
Calls: SELDEV
Description: Output a character to the punch device.

PUNCH:

<table>
<thead>
<tr>
<th>Location</th>
<th>Opcode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0210</td>
<td>215202</td>
<td>LXI H,@PUNCH</td>
</tr>
<tr>
<td>0213</td>
<td>3A0300</td>
<td>LDA IOBYTE</td>
</tr>
<tr>
<td>0216</td>
<td>0F</td>
<td>RRC</td>
</tr>
<tr>
<td>0217</td>
<td>0F</td>
<td>RRC</td>
</tr>
<tr>
<td>0218</td>
<td>0F</td>
<td>RRC</td>
</tr>
<tr>
<td>0219</td>
<td>C37202</td>
<td>JMP SELDEV</td>
</tr>
</tbody>
</table>

; Base address for the Punch devices
; Using the IO Byte / 8

page
FUNCTION READER

Inputs:  None
Outputs: A: character from punch device
Destroys: A,D,E,F,H,L
Calls: SELDEV
Description: Input the character from the reader device.

READER:

021C 215A02 LXI H,@READER ; Base address of the Reader group
021F 3A0300 LDA IOBYTE ; Use the IO Byte once more
0222 0F RRC ; /2
0223 C37202 JMP SELDEV ;
FUNCTION LISTST

Inputs: None
Outputs: A: 00 => ready
         A: FF => not ready
Destroys: A, D, E, F, H, L
Calls: SLEDEV

Description: Return the status of the listing device.

LISTST:
LXI H, @LISTST
LDA 00
RLC RLC RLC
JMP page

; Read the Base address for the List devices
; Using the IO Byte calculate the jump address

0226 31E302
0229 3A3300
022C 07
022D 07
022E 07
022F C37202
022F C37202
FUNCTION: SELDEV

Inputs:  A:=number .... *2 for address width
         H:=group

Outputs: None

Destroys: A,D,E,F,H,L

Calls:   SRSER0,SRSER1,STSER0,STSER1,STPARL,
         RDRST,READER,LIST,
         INSER0,INSER1,OUTSER0,OUTSER1,OUTPARL,
         S1S$ETX,S1S$XON,S1$ETX,S1$XON

Description: According to the state of the "IOBYTE", the logical device is assigned to the physical device.

Jump addresses for the different routines

@CONST:

0232  AB02  DW  SRSER0  ;
0234  B302  DW  SRSER1  ;
0236  7D02  DW  RDRST  ;
0238  B302  DW  SRSER1  ;

@CONIN:

023A  D302  DW  INSER0  ;
023C  DF02  DW  INSER1  ;
023E  1C02  DW  READER  ;
0240  DF02  DW  INSER1  ;

@CONOUT:

0242  EB02  DW  OUTSER0  ;
0244  F602  DW  OUTSER1  ;
0246  0402  DW  LIST  ;
0248  F602  DW  OUTSER1  ;

@LIST:

024A  EB02  DW  OUTSER0  ;
024C  1603  DW  S1$ETX  ;
Copyright 1988 UCT ADAPTED

Page 30
FUNCTION RDRST

Inputs: None

Outputs: A: 00 => ready
A: FF => not ready

 Destroys: A,D,E,F,H,L

Calls: SELDEV

Description: Return the status of the reader device.

RDRST:

027D 216A02 LXI H,@RDRST ; If the Card Reader is the console device
0280 3A0300 LDA IOBYTE ; calculate the jump again
0283 0F RRC
0284 C37202 JMP SELDEV ;

page
FUNCTION: BOOTCOM

Inputs: None

Outputs: None

Destroys: A

Calls: None

Description: Initialise the serial and parallel communications ports as well as the IOBYTE.

\[\text{FUNCTION: BOOTCOM}\]

\[\text{Outputs: None}\]

\[\text{Destroys: A}\]

\[\text{Calls: None}\]

\[\text{Description: Initialise the serial and parallel communications ports as well as the IOBYTE.}\]

\[\text{Initialise the serial and parallel communications ports as well as the IOBYTE.}\]

\[\text{The initial IO Byte:}\]

\[\text{CON:=TTY:}\]

\[\text{RDR:=PTR:}\]

\[\text{PUN:=PTP:}\]

\[\text{LST:=LPT:}\]

\[\text{If there is no data}\]

\[\text{BOOTCOM:}\]

\[\text{Reset the USART}\]

\[\text{Set up the mode}\]
CP/M RMAC ASSEM 1.1
#032 UCT BIOS ADAPTED FOR MASS STORAGE CARD ACCESS

028F D301 OUT SER0$C ;
0291 D305 OUT SER1$C ;
0293 3E37 MVI A,CMD51 ; And then the Command mode
0295 D301 OUT SER0$C ;
0297 D305 OUT SER1$C ;
0299 DB00 IN SER0$D ; Read it
029B DB04 IN SER1$D ;
029D 3E81 MVI A,MODE55 ; Set up the Parallel port
029F D30F OUT PARL$M ;
02A1 3EFF MVI A,NODATA ;
02A3 D30D OUT PARL$B ;
02A5 3E94 MVI A,INTIOBY ; Set the IO Byte to the initial value
02A7 320300 STA IOBYTE ;
02AA C9 RET ;

page
FUNCTION SRSRØ

Inputs: None

Outputs: A: FF => ready
          A: 00 => not ready

Destroys: A,F

Calls: None

Description: Returns the receiving status of the serial-A port.

SRSRØ:

02AB DB01 IN SER0$C ; Read the status
02AD E602 ANI RXRDY ; Ready?
02AF C8 RZ ; No - Return with A = 0
02B0 F6FF ORI OFFH ; Yes - Return with A = FF
02B2 C9 RET
page
FUNCTION SRSERl

Inputs: None

Outputs: A: FF => ready
          A: 00 => not ready

Destroys: A,F

Calls: None

Description: Returns the receiving status of the serial-B port.

SRSERl:

02B3 DB05    IN    SERl$C    ; Read the Status
02B5 E602    ANI    RXRDY    ; Ready?
02B7 C8      RZ
0288 F6FF    ORI    0FFH    ; no - set A = 0 and return
02BA C9      RET    0FFH    ; yes - set A = FF and return
FUNCTION STSERØ

Inputs: None

Outputs: A:=FF => ready
A:=00 => not ready

Destroys: A,F

Calls: None

Description: Returns the transmit status of the serial-A port.
FUNCTION STSER1
Inputs: None
Outputs: A: FF => ready
A: 0 => not ready
Destroys: A,F
Calls: None
Description: Returns the transmit status of the serial-B port.

STSER1:

; Read the Status
; Ready to Send?
; no - Return with A = 0
; yes - Return with A = FF

02C3 DB05 IN SER1$C
02C5 E601 ANI TXRDY
02C7 C8 RZ
02C8 F6FF ORI OFFH
02CA C9 RET
page
FUNCTION STPARL

Inputs: None
Outputs: A: FF => ready
A: 00 => not ready
Destroys: A,F
Calls: None
Description: Returns the transmit status of the parallel port.

STPARL:

02CB DB0E
02CD E602
02CF 3D
02D0 F8
02D1 AF
02D2 C9

IN PARL$C ; Read the parallel port status
ANI PTXRDY ; Ready to send?
DCR A ; yes - set A =0FFH and return
RM
XRA A ; no - set A =00H and return
RET ;
;----------------------------------------------------------
;  FUNCTION INSERØ
;  Inputs:   None
;  Outputs:  A: character
;  Destroys: A,F
;  Calls:    None
;  Description: Waits until a character is ready and returns it
;               from the serial-A port.
;----------------------------------------------------------

Ø2D3 DB01
Ø2D5 B602
Ø2D7 CAD302
Ø2DA DB00
Ø2DC B67F
Ø2DE C9

IN   SERØ$C   ; Read the Status
ANI   RXRDY   ; is it ready to transmit
JZ    INSERØ   ; no - loop
IN   SERØ$D   ; yes - Read the input register
ANI   NOPAR   ; and remove the parity
RET

; page
FUNCTION INSERl

Inputs: None
Outputs: A:=character
Destroys: A,F
Calls: None
Description: Waits until a character is ready and returns it from the serial-B port.

INSERl:

IN SERl$C ; Read the status register
ANI RXRDY ; and wait until it is ready
JZ INSERl ;
IN SERl$D ; then read the character and
ANI NOPAR ; remove its parity bit.
RET ;

page
FUNCTION: OUTSER0

Inputs: C: character
Outputs: None
Destroys: A,F
Calls: None
Description: Outputs a character to the serial-A port when it is ready.

OUTSER0:

02E8 DB01 IN SER0$C ; Read the Status register
02ED E601 ANI TXRDY ; and wait until it is ready to send
02EF CAEB02 JZ OUTSER0 ;
02F2 79 MOV A,C ; Then send the character
02F3 D300 OUT SER0$D ;
02F5 C9 RET ;

page
FUNCTION: OUTSER1

Inputs:  C: character

Outputs: None

Destroy: A, F

Calls: None

Description: Outputs a character to the serial-B port when it is ready.

OUTSER1:

02F6 DB05  IN  SER1$C  ; Read the status register
02F8 E601  ANI  TXRDY  ; and wait until it is ready to send
02FA CAF602  JZ  OUTSER1  ;
02FD 79  MOV  A,C  ; then send the character
02FE D304  OUT  SER1$D  ;
0300 C9  RET  ;

page
FUNCTION: OUTPARL

Inputs:  C: character
Outputs: None
Destroys: A,F
Calls: None
Description: Outputs a character to the parallel port when it is ready. # CENTRONICS FORMAT #

0080 = NOSTB EQU 10000000B ; No strobe bit
007F = STROBE EQU 01111111B ; The strobe character

OUTPARL:
0301 DB0E IN PARL$C ; Read the status register
0303 E602 ANI PTXRDY ; Wait until the transmit bit is set
0305 C20103 JNZ OUTPARL ;
0308 3B80 MVI A,NOSTB ; Send the character to the port
030A B1 ORA C ; with no strobe initially
030B D30D OUT PARL$B ; And then with the strobe
030D E67F ANI STROBE
030F D30D OUT PARL$B ;
0311 F680 ORI NOSTB ; Reset the strobe line
0313 D30D OUT PARL$B ;
0315 C9 RET ;
FUNCTION: Sl$ETX  
Inputs: C: character  
Outputs: None  
Destroys: A,F,H,L  
Calls: SlS$ETX  
Description: Output a character to the serial-B port according  
to the ETX/ACK protocol. # DIABLO PRINTER #

Sl$ETX:

0316 CD2E03 CALL S1S$ETX ; Get the status of Port B using ETX/ACK
0319 CA1603 JZ S1$ETX ; Wait until ready
031C 79 MOV A,C ; then send it
031D D304 OUT SER1$D ;
031F 218803 LXI H,ETX$CNT ; Counter-1
0322 35 DCR M ;
0323 C9 RET ;

page

# diagonal line

DIABLO PRINTER #
FUNCTION: $044

Inputs: C: character
Outputs: None
Destroys: A,F,H,L
Calls: $044

Description: A character is sent to the serial-B port when it is ready according to the X-OFF/X-ON protocol.

$044:
CALL $044
JZ $044
MOV A,C
OUT SER1$D
RET
FUNCTION SLS$ETX

Inputs: None

Outputs: A: $00 => not ready
         $0F => not L
         $FF => ready

Description: The status of the serial-B port is returned according to the ETX/ACK protocol. The values of ETXCNT: $0 = Send ETX if ready
              $FF = Is port ready
              $50 = Receive ACK if ready

Calls: None

The status of the serial-B port is returned according to the ETX/ACK protocol.

ETX EQU 03H

ACK EQU 06H

SLS$ETX:
LXI H,ETXCNT
XRA A
CMP M
JNZ CASE1
If it is zero then send ETX

IN  SER1$C
ANI TXRDY
RZ
Return with A = $0 means not ready

DCR M ; Change the count to ETX sent
MVI A,ETX
OUT SER1$D
XRA A ; Set A=0 as not yet ready
RET
CASE1:
JP CASEELSE

CASEELSE:

CASE1:

0003 = ETX
0006 = ACK

0328 218803
0331 AF
0332 BE
0333 C24303
0334 E601
0335 DB05
0336 C24303
0337 0300
0338 0300
0339 0300
033A 0300
033B 0300
033C 0300
033D 0300
033E 0300
033F 0300
0340 0300
0341 C9
0342 F25703

0345 0300
0346 0300
0347 0300
0348 0300
0349 0300
034A 0300
034B 0300
034C 0300
034D 0300
034E 0300
034F 0300
0350 0300
0351 0300
0352 0300
0353 0300
0354 0300
0355 0300
0356 0300
0357 0300
0358 0300
0359 0300
035A 0300
035B 0300
035C 0300
035D 0300
035E 0300
035F 0300
0360 0300
0361 0300
0362 0300
0363 0300
0364 0300
0365 0300
0366 0300
0367 0300
0368 0300
0369 0300
036A 0300
036B 0300
036C 0300
036D 0300
036E 0300
036F 0300
0370 0300
0371 0300
0372 0300
0373 0300
0374 0300
0375 0300
0376 0300
0377 0300
0378 0300
0379 0300
037A 0300
037B 0300
037C 0300
037D 0300
037E 0300
037F 0300
0380 0300
0381 0300
0382 0300
0383 0300
0384 0300
0385 0300
0386 0300
0387 0300
0388 0300
0389 0300
038A 0300
038B 0300
038C 0300
038D 0300
038E 0300
038F 0300
0390 0300
0391 0300
0392 0300
0393 0300
0394 0300
0395 0300
0396 0300
0397 0300
0398 0300
0399 0300
039A 0300
039B 0300
039C 0300
039D 0300
039E 0300
039F 0300
03A0 0300
03A1 0300
03A2 0300
03A3 0300
03A4 0300
03A5 0300
03A6 0300
03A7 0300
03A8 0300
03A9 0300
03AA 0300
03AB 0300
03AC 0300
03AD 0300
03AE 0300
03AF 0300
03B0 0300
03B1 0300
03B2 0300
03B3 0300
03B4 0300
03B5 0300
03B6 0300
03B7 0300
03B8 0300
03B9 0300
03BA 0300
03BB 0300
03BC 0300
03BD 0300
03BE 0300
03BF 0300
03C0 0300
03C1 0300
03C2 0300
03C3 0300
03C4 0300
03C5 0300
03C6 0300
03C7 0300
03C8 0300
03C9 0300
03CA 0300
03CB 0300
03CC 0300
03CD 0300
03CE 0300
03CF 0300
03D0 0300
03D1 0300
03D2 0300
03D3 0300
03D4 0300
03D5 0300
03D6 0300
03D7 0300
03D8 0300
03D9 0300
03DA 0300
03DB 0300
03DC 0300
03DD 0300
03DE 0300
03DF 0300
03E0 0300
03E1 0300
03E2 0300
03E3 0300
03E4 0300
03E5 0300
03E6 0300
03E7 0300
03E8 0300
03E9 0300
03EA 0300
03EB 0300
03EC 0300
03ED 0300
03EE 0300
03EF 0300
03F0 0300
03F1 0300
03F2 0300
03F3 0300
03F4 0300
03F5 0300
03F6 0300
03F7 0300
03F8 0300
03F9 0300
03FA 0300
03FB 0300
03FC 0300
03FD 0300
03FE 0300
03FF 0300
CUIT BIOS ADAPTED FOR MASS STORAGE CARD ACCESS

0345 DB05 IN SERI$C ; ACK received?
0347 B602 ANI RXRDY ;
0349 C8 RZ ; Nothing at the port, return with A = 0
034A DB04 IN SERI$D ; Is the character an ACK?
034C E67F ANI NOPAR ;
034E FE06 CPI ACK ;
0350 CA5503 JZ ACKREC ; yes - Adjust counter and see if ready to transmit
0353 AF XRA A ; no - Set A = 0 and return
0354 C9 RET ;

ACKREC:
0355 3632 MVI M,50 ; To show that ACK has been received

CASEELSE:
0357 DB05 IN SERI$C ; Read the status register
0359 B601 ANI TXRDY ; Ready to send?
035B C8 RZ ; no - Set A = 0 and return
035C FEFF ORI 0FFH ; yes - Set A = 0FFH and Return
035E C9 RET ;

page
FUNCTION S1S$XON

Inputs: None
Outputs: A: FF => ready
A: 00 => not ready
Destroys: A,F,H,L
Calls: None
Description: The status of the serial-B port is returned according to the X-ON/X-OFF protocol.

;------------------------------------------------------------------------------

XON EQU 11H ; ASCII values
XOFF EQU 13H

; S1S$XON:
035F 218703 LXI H,XONFLAG ; Set pointer to the flag
0362 DB05 IN SER1$C ; Read the status
0364 E602 ANI RXRDY ; Receive ready?
0366 CA7E03 JZ NOINPUT ; no - Try for output ready
0369 DB04 IN SER1$D ; yes - Read the data
036B E67F ANI NOPAR ; removing the parity bit
036D FE11 CPI XON ; Is it an X-ON character?
036F C27703 JNZ CNTL$S ; no - Try for X-OFF
0372 36FF MVI M,0FFH ; yes - Set flag
0374 C37E03 JMP NOINPUT ;

; CNTL$S:
0377 FE13 CPI XOFF ; Is it a X-OFF?
0379 C27E03 JNZ NOINPUT ; no - then ignore it
037C 3600 MVI M,0 ; yes - Reset the flag

; NOINPUT:
037E DB05 IN SER1$C ; Read the Status byte
CP/M RMAC ASSEM 1.1

0380 E601 ANI TXRDY ; Is the Ready bit clear
0382 A6 ANA M ; or the Flag = 0?
0383 C8 RZ ; yes - Return with A=0
0384 F6FF ORI 0FFH ; no - Set A=0FFH and return
0386 C9 RET

page
SECTION 6 - Variables of the Communications Package

XONFLAG DB 0FFH ; FF=X-ON received
; 0=X-OFF received
ETXCNT DB 50 ; 0=Transmit ETX
; 50=ACK received
; FF=Wait for Port to ready

SECTION 7 - Variables of the Disk Package and DPBlock

SEKDSK DB 0 ; Selected disk
ADROFF DS 1 ; Gives the address offset for 128 byte pages
EPOSET DS 1 ; Saves the first EPROM number
EPNUMS DS 1 ; Saves the number of the present EPROM
DMAADR DW 0080H ; DMA address storage space

END
APPENDIX B

The Cold Start Loader for the UCT Micro-computers
;==============================================================================

; EQUATES

;===================================~=====================================

0D0A = LF EQU 0D0AH ; Carriage return line feed
1000 = STRT EQU 1000H ; The start address
F300 = CPMBOOT EQU 0F300H ; The entry point for BIOS
DD00 = NEWPOS EQU 0DD00H ; Start of BDOS
1A80 = LEN EQU 1A80H ; Length of the BDOS, BIOS and CCP
DD00 = SAVSP EQU 0DD00H ; Save space for transfer of data
0023 = ENUM EQU 23H ; Eprom Number address
0022 = EADHI EQU 22H ; High Address on the Eprom selected
0021 = EADLO EQU 21H ; Low Address on the Eprom selected
0020 = EDATA EQU 20H ; Data from the Eprom selected
0010 = MAXEP EQU 16 ; Maximum number of EPROMs on the Mass storage card
F03D = OUTMSG EQU 0F03DH ; SABus string output routine

page
CP/M RMAC ASSEM 1.1  #003  COLD START LOADER FOR UCT KITS

;=============================================================================
; LOCAL DATA SEGMENT
;=============================================================================

DSEG

EPNUM:
0000
SAVED:
0001
0003
STAC:

DSEG

; Save the present EPROM number
DS 1

; Save space for CCP address
DS 2

; Stack space
DS 50

page
START OF MAIN PROGRAM

CSEG

0000 CD3D00 CALL WHICHEP ; Find the start of the 'Eprom disk'
0003 3C INR A ; An error return?
0004 CA1D00 JZ NODIR ;
0007 3D DCR A
0008 320000 STA EPNUM ; Save the current eprom number
000B 313500 LXI SP,STAC ; Set up the stack pointer
000E 116600 LXI D,OLDPOS ; DE = Position of BIOS and BDOS
0011 010000 LXI B,NEWPOS ; BC = The new position
0014 21801A LXI H,LEN ; HL = Length of the data
0017 2C5000 CALL TXLOOP ; Move the data
001A C300F3 JMP CPMBOOT ; Boot up using the BIOS

NODIR:

001D 212400 LXI H,ERRMSGF ; Send an error message
0020 CD3DF0 CALL OUTMSG ;
0023 76 HLT

ERRMSG:

0024 0A0D0774E DB 0AH,0DH,07H,07H,'No Directory found',0DH,0AH,00H
FUNCTION: WHICHEP

; INPUTS: None
; OUTPUTS: A - Start of the Eprom Disk less one
; CALLS: None

; DESCRIPTION:
; WHICHEP reads the first byte on each EPROM on the mass storage card
; until it finds a 00H. This must be the start of the directory and so
; it returns the EPROM number.

;-----------------------------------------------------------------------------

WHICHEP:

003D 0E00 MVI C,0 ; Try the first EPROM
003F 79 MOV A,C

TRYNEXT:

0040 D323 OUT ENUM ;
0042 3E00 MVI A,0 ; And set the addresses to zero
0044 D322 OUT EADHI ;
0046 D321 OUT EADLO ;
0048 DB20 IN EDATA ; Read the data
004A FE00 CPI 0 ; Is it zero
004C CA5900 JZ FOUND ; Goodie
004F 0C INR C ; End of the EPROMs ?
0050 79 MOV A,C ;
0051 FE10 CPI MAXEP ;
0053 C24000 JNZ TRYNEXT ;
0056 3EFF MVI A,0FFH ; Error return
0058 C9 RET

FOUND:

0059 79 MOV A,C ;
005A C9 RET ;
FUNCTION: TXLOOP

INPUTS:  
- BC - points at the new position
- DE - containing the pointer pointing at the data
- HL - Contains the length to be transferred

OUTPUTS:  
- Data to the new position

CALLS: None

DESCRIPTION:

This routine reads the data from the old position in memory and puts it in the new data position.

TXLOOP:

005B 1A  LDAX  D  ; Read the data from the old position
005C 02  STAX  B  ; Put it in the new position
005D 13  .INX  D  ; Point at the new address
005E 03  .INX  B  ;
005F 2B  DCX  H  ; Check the length
0060 7C  MOV  A,H  ;
0061 B5  ORA  L  ;
0062 C25B00  JNZ  TXLOOP  ;
0065 C9  RET  ; Finished

OLDPOS:

***************INCLUDE CCP, BDOS AND BIOS******************
APPENDIX C

The EPROM Management System
EPROM Management System

(* 15/01/85 *)
(* *************************************************************)
(* EPROM Management System *)
(* ======================= *)
(* This program has a number of sections to it: *)
(* 1. EPROM map display *)
(* 2. File addition *)
(* 3. File deletion *)
(* 1. The EPROM display is a diagrammatic display of the programs as they *)
(* are found on the EPROM Card in the UCT SABus kits. *)
(* 2. File addition enables the user to add a file to the EPROM card and *)
(* then tells the user which EPROMs must be reprogrammed. *)
(* 3. File deletion lets the user take a file off the mass storage card *)
(* and so make space for other programs. *)
(* ******************************************************* *)

PROGRAM EMS;

CONST  CPM_BLK = 128;
       MAXENT = 128;

TYPE  DRIVES = $0..$0F;
       FCB = RECORD
           DRIVE : DRIVES;
           FILENAME : PACKED ARRAY[1..11] OF CHAR;
           SCRATCH1,SCRATCH2,SCRATCH3 : BYTE;
           NUM BLKS : BYTE;
           BLOC_LOC : ARRAY[1..16] OF BYTE;
       END;
EPROM Management System

FYLE = FILE OF BYTE;
FYL = FILE OF FCB;

VAR
    INFILE : FYL;
    BUF : ARRAY[1..MAXENT] OF FCB;
    BUFR : ARRAY[1..MAXENT] OF BYTE;
    PLOT : ARRAY[1..MAXENT] OF BOOLEAN;
    OFFSET,RESULT,MAXENTRY : INTEGER;
    ERROR : BOOLEAN;
    ORDER : ARRAY[1..MAXENT] OF INTEGER;
    N : FYLE;
    E : FYLE;
    ANS,NEW_FILE,EP_FILE : STRING;
    RESPONSE : CHAR;
EXTERNAL PROCEDURE @HLT;

FUNCTION ERRCHK : BOOLEAN;
(* This procedure checks to see if there were any errors in the SEEKREAD call*)
VAR
  ERR : BOOLEAN;
BEGIN
  ERROR := FALSE;
  IF IORESULT <> 0 THEN
    ERROR := TRUE;
    ERRCHK := ERR;
END;

(*---------------------------------------------------------------------------*)
EPROM Management System

FUNCTION READ_DIR(VAR DIR_FYL : FYL) : INTEGER;
(* Reads the Records from the EPDIR file to a buffer *)
VAR
  I    : INTEGER;
  ERROR : BOOLEAN;
BEGIN
  I := 0;
  REPEAT
    SEEKREAD(DIR_FYL,I);
    ERROR := ERRCHECK;
    I := I+1;
    IF I < MAXENT THEN
      BEGIN
        IF NOT ERROR THEN
          BUF[I] := DIR_FYL;
        END
      ELSE
        ERROR := TRUE;
    END
  UNTIL ERROR;
  I := 0;
  REPEAT
    I := I+1;
  UNTIL BUF[I].DRIVE = $E5;
  READ_DIR := (I-1);
END;

(*---------------------------------------------------------------------------*)
EPROM Management System

PROCEDURE SORT_ENTRIES(TOP_ENT : INTEGER);
(* Sorts the entries in an array pointer so that the records will be in *)
(* order when printed.*)
VAR   I,J,TEMP : INTEGER;
X : FCB;
BEGIN
   FOR I := 1 TO TOP_ENT DO
      BEGIN
         FOR J := I+1 TO TOP_ENT DO
            BEGIN
                  BEGIN
                     X := BUF[J];
                     BUF[J] := BUF[I];
                     BUF[I] := X;
                  END;
            END;
      END;
END;
PROCEDURE SET UP CON;
(* Outputs the basic headings to the console.*)
BEGIN
    WRITE('£-------~--£------------------------------');
    WRITELN('-------------------------£');
    WRITE('£ EPROM £ 2k Block Number £');
    WRITELN('£');
    WRITE('£NUMBER £ 0 £ 1 £');
    WRITELN(' 2 £ 3 £');
    WRITE('£----------£-------------£-------------£');
    WRITELN('-------------£-------------£ 1 £');
    WRITE(' £ 0 £');
END;
(*-------------------------------------------------------------------------*)
EPROM Management System

PROCEDURE FILL_BLOCK(X,Y : INTEGER);
(* Fills the empty blocks*)
VAR I : INTEGER;
BEGIN
  IF X < 64 THEN
  BEGIN
    WRITE(' --- £ ');
    FOR I := (X+1) TO Y DO BEGIN
      IF I MOD 4 = 0 THEN BEGIN
        WRITELN;
        IF I > 39 THEN WRITE('£ ,TRUNC(I/4), £ --- £ ')
        ELSE WRITE('£ ,TRUNC(I/4), £ --- £ ')
      END ELSE WRITE('£ ,TRUNC(I/4), £ --- £ ')
    END;
  END;
  WRITE(' --- £ ');
END;
(*---------------------------------------------------------------------------*)
EPROM Management System

PROCEDURE ENUM(X : INTEGER);
(*Puts on the EPROM number in the Map*)
BEGIN
  IF X < 63 THEN
  BEGIN
    IF X MOD 4 = 0 THEN
    BEGIN
      WRITELN;
      IF X > 39 THEN
        WRITE('£',TRUNC(X/4),' £')
      ELSE
        WRITE('£',TRUNC(X/4),' £')
    END;
  END;
END;
(*---------------------------------------------------------------------------*)
FUNCTION FIND TOP : INTEGER;
(* Finds the highest block written to on the EPROM card*)
VAR
   I, J, X : INTEGER;
BEGIN
  X := 0;
  FOR I := 1 TO 64 DO
    BEGIN
      FOR J := 1 TO 16 DO
        BEGIN
          IF (BUF[I].BLOC_LOC[J] > X) AND (BUF[I].BLOC_LOC[J] <> $ES) THEN
            X := BUF[I].BLOC_LOC[J];
          END;
        END;
    END;
  FIND_TOP := X;
END;

(*---------------------------------------------------------------*)
EPROM Management System

PROCEDURE DIREC( BASE : INTEGER);
(*Fills up the map until OFFSET and then writes 'DIRECTORY'*)
BEGIN
  IF BASE < 63 THEN
    BEGIN
      IF BASE > 4 THEN
        BEGIN
          FILL_BLOCK(0,BASE-2);
          ENUM(BASE-1);
        END;
        WRITE(' DIRECTORY £ ');
      END;
    END;
END;

(*---------------------------------------------------------------------------*)
**EPROM Management System**

PROCEDURE DISPLAY_MAP(OFF TOP_ENT : INTEGER);
(* Does the filling in of the EPROM map using the EPDIR file.*)

VAR O, I, J, K, BLOCK_COUNT : INTEGER;

BEGIN
  O := OFF*4;
  BLOCK_COUNT := O+1;
  DIREC(BLOCK_COUNT);
  REPEAT
    I := 1;
    REPEAT
      J := 1;
      REPEAT
        J := J+1;
        UNTIL (J = 17) OR (BUF[I].BLOC_LOC[(J-1)] = (BLOCK_COUNT - 0));
        I := I+1;
      UNTIL (I = (TOP_ENT+1)) OR (BUF[(I-1)].BLOC_LOC[(J-1)] = (BLOCK_COUNT-0));
      IF (BUF[(I-1)].BLOC_LOC[(J-1)] = (BLOCK_COUNT-0)) THEN
        BEGIN
          FOR K := 1 TO 11 DO
            BEGIN
              WRITE(BUF[(I-1)].FILENAME[K]);
              END;
              WRITE(' E ');
            END
          ELSE
            FILL_BLOCK(BLOCK_COUNT, BLOCK_COUNT);
            BLOCK_COUNT := BLOCK_COUNT + 1;
            ENUM(BLOCK_COUNT);
          UNTIL (BLOCK_COUNT = 64);
          WRITELN;
        END;
      END;

(*---------------------------------------------------------------*)
EPROM Management System

PROCEDURE END_DISPLAY;
(*Puts the bottom line on the map*)
BEGIN
  WRITE('£----------£-------------£-------------£');
  WRITELN('-------------£-------------£');
  WRITELN('-----------------£-----------------£');
END;
PROCEDURE KILL_IT(NAME, NAME1 : STRING; TOP_ENT : INTEGER);
(* Removes the directory entry.*)
VAR
   EXTENT, I, J : INTEGER;
   DONE, MATCH : BOOLEAN;
BEGIN
   DONE := FALSE;
   I := 1;
   EXTENT := 0;
   REPEAT
      MATCH := TRUE;
      J := I;
      REPEAT
         IF BUF[I].FILENAME[J] <> NAME[J] THEN
            MATCH := FALSE;
            J := J+1;
         UNTIL (NOT MATCH) OR (J = 12);
         I := I+1;
      UNTIL (MATCH) OR (I = (TOP_ENT+1));
      IF MATCH THEN
         BEGIN
            FOR J := (I-1) TO TOP_ENT DO
               BUF[J] := BUF[(J+1)];
            DONE := TRUE;
            EXTENT := EXTENT + 1;
         END;
         I := I-1;
      UNTIL I = TOP_ENT;
END;
EPROM Management System

IF NOT DONE THEN
BEGIN
  WRITELN;
  WRITELN('THERE IS NO ENTRY: ', NAME);
  EXIT;
END
ELSE
BEGIN
  WRITELN;
  WRITELN('RE-PROGRAM : EDPDIR.COM');
  MAXENTRY := MAXENTRY - EXTENT;
END;
END;
EPROM Management System

(************************************************************************************
(* UTILITY PROCEDURES FOR FILE ADDITION *)
(*******************************************************************************************)

FUNCTION FILE_SIZE(VAR X : FYLE) : INTEGER;
(*This function calculates the size of the file specified by X*)
VAR
    ENDED : BOOLEAN;
    RES,I : INTEGER;
BEGIN
    ENDED := FALSE;
    I := 0;
    REPEAT
        BLOCKREAD(X,BUFR,RES,CPM_BLK,I);
        IF Res = 0 THEN
            I := I+1
        ELSE
            ENDED := TRUE;
        UNTIL ENDED;
    FILE_SIZE := I;
END;

(*---------------------------------------------------------------------------*)
EPROM Management System

FUNCTION CONV_SIZE(Y : INTEGER) : INTEGER;
(*Converts Y (a 128 byte block count) to a 2k block count*)
BEGIN
  IF (Y MOD 16) = 0 THEN
    CONV_SIZE := TRUNC(Y/16)
  ELSE
    CONV_SIZE := TRUNC(Y/16)+1;
  END;

(*---------------------------------------------------------------------------*)
FUNCTION CHECK_SPACE(VAR DIR : FYL; NUM,OFF : INTEGER) : INTEGER;
(*Gives the position at which the new program must be loaded or 64 if there *)
(* is not enough space.*)
VAR
  TOTAL_ENT,J,K,L : INTEGER;
  NO : BOOLEAN;
BEGIN
  TOTAL_ENT := READ_DIR(DIR);;
  SORT_ENTRIES(TOTAL_ENT);
  FOR J := 1 TO 64 DO
    PLOT[J] := FALSE;
  FOR J := 1 TO TOTAL_ENT DO
    FOR K := 1 TO 16 DO
      PLOT[BUF[J],BLOC_LOC[K]] := TRUE;
    FOR J := (65-(OFF*4)) TO 64 DO
      PLOT[J] := TRUE;
  WRITELN;
  WRITE('Enter eproms NOT available ');
  WRITE('(actual positions, -1<cr> if all available)');
  WRITE(' ');
**EPROM Management System**

REPEAT
   READ(K);
   K := K-OFF;
   IF (K < 17) AND (K > -1) THEN
      FOR J := (K*4) TO ((K*4)+3) DO
         PLOT[J] := TRUE;
   UNTIL EOLN;
   WRITELN('THANKS');
   WRITELN;
   K := 0;
   FOR J := 1 TO 64 DO
      IF NOT PLOT[J] THEN
         K := K+1;
      IF NUM < K THEN
         BEGIN
            L := 0;
            REPEAT
               L := L + 1;
               UNTIL NOT PLOT[L];
            L := L - 1;
         END
      ELSE
         BEGIN
            L := 64;
            CHECK_SPACE := L;
         END;
   END;
(*---------------------------------------------------------------------------*)
EPROM Management System

FUNCTION SET_UP_ENTRY(TOP_ENT,BLK_1,S,S_2K : INTEGER) : INTEGER;
(*Sets up the entry in the CP/M format*)
VAR
   I,J,K,L,M,N,P,EXT : INTEGER;
   COPY_NF : STRING;
BEGIN
   I := TOP_ENT+1;
   COPY_NF := NEW_FILE;
   DELETE(COPY_NF,1,POS(';',COPY_NF));
   J := POS('.',COPY_NF);
   DELETE(COPY_NF,J,1);
   IF LENGTH(COPY_NF) < 11 THEN
      REPEAT
         INSERT(' ',COPY_NF,J);
      UNTIL LENGTH(COPY_NF) = 11;
   EXT := (TRUNC(S_2K/8)+1);
   N := EXT;
   P := S_2K;
   FOR J := 1 TO EXT DO
      BEGIN
         N := N-1;
         L := I+(J-1);
         BUF[L].DRIVE := $0;
         FOR K := 1 TO 11 DO
            BUF[L].FILENAME[K] := COPY_NF[K];
         BUF[L].SCRATCH1 := J-1;
         BUF[L].SCRATCH2 := 0;
         BUF[L].SCRATCH3 := 0;
      END;
END;
IF N = 0 THEN
    BUF[L].NUM_BLKS := S
ELSE
    BEGIN
    BUF[L].NUM_BLKS := $80;
    S := S - $81;
    END;
M := 0;
REPEAT
    M := M + 1;
UNTIL (M > 64) OR (NOT PLOT[M]);
IF N = 0 THEN
    BEGIN
    FOR K := 1 TO P DO
    BEGIN
    BUF[L].BLOC_LOC[2*K] := $00;
    PLOT[M] := TRUE;
    REPEAT
    M := M + 1;
    UNTIL (NOT PLOT[M]) OR (M > 64);
    END;
    IF BUF[L].BLOC_LOC[16] = $E5 THEN
    FOR K := (P+1) TO 8 DO
    BEGIN
    BUF[L].BLOC_LOC[2*K-1] := $00;
    BUF[L].BLOC_LOC[2*K] := $00;
    END;
    END;
ELSE
BEGIN
FOR K := 1 TO 8 DO
BEGIN
  BUF[L].BLOC_LOC[2*K] := $00;
  PLOT[M] := TRUE;
  REPEAT
    M := M+1;
    UNTIL (NOT PLOT[M]) OR (M > 64);
  END;
END;
END;
P := P-8;
END;
SET_UP_ENTRY := EXT;
END;

>(*---------------------------------------------------------------------------*)
EPROM Management System

PROCEDURE UPDATE_DIR(TOP_ENT, X : INTEGER);
(*Updates the EPDIR.COM program.*)
VAR I : INTEGER;
BEGIN
  FOR I := 1 TO X DO
  BEGIN
    INFILE := BUF[(TOP_ENT+I)];
    SEEKWRITE(INFILE, (TOP_ENT+I-1));
  END;
  IF IORESULT <> 0 THEN
  BEGIN
    WRITELN('ERROR IN EXTENDING EPDIR.COM');
    EXIT;
  END;
END;

(*---------------------------------------------------------------------------*)
EPROM Management System

PROCEDURE WHAT_FILE(BLK I : INTEGER);
(*Opens the EPn.COM on the basis of I.*)
BEGIN
CASE TRUNC(BLK I/4) OF
  0 : EP_FILE := 'EPDIR.COM';
END;
END;
(*---------------------------------------------------------------*)
EPROM Management System

PROCEDURE NEW_EPROM(VAR INF: FILE; TOP_ENT, EX, S, S_2K: INTEGER);
(*Creates the EPn.COM programs and saves them on disk*)
VAR
  X, I, J, K, BLK, RES, C_BLK: INTEGER;
  TRANS_BUF: ARRAY[1..CPM_BLK] OF BYTE;
  OEFILE, EFILE: STRING;
  S_128, M: BYTE;
BEGIN
  WRITELN;
  WRITELN('BURN THE FOLLOWING PROGRAMS INTO EPROMS:');
  WRITELN('EPDIR.COM');
  RESET(INF);
  X := 0;
  FOR J := 1 TO EX DO
    BEGIN
      I := 1;
      BLK := BUF[TOP_ENT+J].BLOC_LOC[I];
      S_128 := BUF[TOP_ENT+J].NUM_BLKS;
      M := 0;
      WHAT_FILE(BLK);
      OEFILE := EP_FILE;
      ASSIGN(E, EP_FILE);
      RESET(E);
      REPEAT
        BLK := BUF[TOP_ENT+J].BLOC_LOC[I];
        WHAT_FILE(BLK);
        EFILE := EP_FILE;
EPROM Management System

IF EFILE <> OFILE THEN
BEGIN
  CLOSE(E, RES);
  IF RES <> 0 THEN
  BEGIN
    WRITELN('ERROR IN CLOSING ', OFILE);
    EXIT;
  END;
  WRITELN(OFILE); (* EPROM to be programmed.*)
  ASSIGN(E, EFILE);
  RESET(E);
  OFILE := EFILE;
END;

C_BLK := (BLK MOD 4) * 16; (* C_BLK has the CP/M block number.*)
REPEAT
  BLOCKREAD(INF, TRANS_BUF, RES, CPM_BLK, X);
  IF RES <> 0 THEN
    M := S_128; (* To show end of data.*)
    IF M < S_128 THEN
      BEGIN
        BLOCKWRITE(E, TRANS_BUF, RES, CPM_BLK, C_BLK);
      END;
      IF RES <> 0 THEN
        BEGIN
          WRITELN('DISK FULL');
          EXIT;
        END;
  END;
  X := X + 1;
  M := M + 1;
  C_BLK := C_BLK + 1;
UNTIL (C_BLK MOD 16 = 0) OR (M > S_128);
EPROM Management System

I := I+2;
UNTIL (BUF[TOP_ENT+J].BLOC_LOC[I] = Ø) OR (I > 16) OR (M > S_128);
CLOSE(E,RES);
WRITELN(EFILE);
END;
END;

(*---------------------------------------------------------------------------*)
FUNCTION THERE(ADD_FILE : STRING; TOP_ENT : INTEGER) : BOOLEAN;
(* Checks to see if the file specified by ADD_FILE is in the directory.*)
VAR
    I, J : INTEGER;
    Y : BOOLEAN;
    A_FILE : STRING;
    REP : CHAR;
BEGIN
    A_FILE := ADD_FILE;
    DELETE(A_FILE,1,POS(';', A_FILE));
    J := POS('.', A_FILE);
    DELETE(A_FILE, J, 1);
    IF LENGTH(A_FILE) < 11 THEN
        REPEAT
            INSERT(' ', A_FILE, J);
            UNTIL LENGTH(A_FILE) = 11;
    I := 1;
    REPEAT
        Y := TRUE;
        J := 1;
        REPEAT
                Y := FALSE;
                J := J+1;
            UNTIL (NOT Y) OR (J = 12);
        I := I+1;
    UNTIL (Y) OR (I = (TOP_ENT+1));
IF Y THEN
BEGIN
  WRITELN('There is another ',ADD_FILE,' in memory already,');
  WRITE('D)elete it or R)eturn to main menu [D/R] ');
  READLN(REP);
  IF REP = 'D' THEN
  BEGIN
    KILL IT(A_FILE,A_FILE, TOP_ENT);
    J := TOP_ENT;
    TOP_ENT := ∅;
    UPDATE_DIR(TOP_ENT, J);
    THERE := TRUE;
  END
ELSE
  THERE := FALSE;
END
ELSE
  THERE := TRUE;
END;
PROCEDURE EPROM_MAP(OFFS : INTEGER);
(*This is the procedure that outputs the map of data in the EPROM card*)
VAR
    RESP : CHAR;
    TOTAL_ENT : INTEGER;
BEGIN
    OPEN(INFILE,'EPDIR.COM',RESULT);
    IF RESULT = 255 THEN
        BEGIN
            WRITELN('NO EPDIR PRESENT ON THIS DISK');
            @HLT;
        END
    ELSE
        BEGIN
            TOTAL_ENT := READ_DIR(INFILE);
            SORT_ENTRIES(TOTAL_ENT);
            SET_UP_CON;
            DISPLAY_MAP(OFFS,TOTAL_ENT);
            END_DISPLAY;
            WRITE('Hit <CR> to continue');
            READLN(RESP);
        END;
END;
(*---------------------------------------------------------------------------*)
PROCEDURE FILE_ADD(TOTAL_ENT: INTEGER);
(*This is the main controlling procedure for file addition.*)
VAR
  CONT: BOOLEAN;
  SIZ_128, SIZ_2K, SPCE, EXT: INTEGER;
BEGIN
  WRITELN;
  WRITELN;
  WRITELN('FILE ADDITION UTILITY');
  WRITELN('================================');
  WRITE('NAME OF NEW FILE TO ADD > ');
  READLN(NEW_FILE);
  CONT := THERE(NEW_FILE, TOTAL_ENT);
  IF CONT THEN BEGIN
    OPEN(N, NEW_FILE, RESULT);
    IF RESULT <> 0 THEN BEGIN
      WRITELN('THERE IS NO ', NEW_FILE);
      EXIT;
    END;
    ASSIGN(INFILE, 'EPDIR.COM');
    RESET(INFILE);
    IF IORESULT <> 0 THEN BEGIN
      WRITELN('THERE IS NO EPDIR.COM ON THIS DISK');
      @HLT;
    END;
  END;
END;
SIZ_128 := FILE_SIZE(N); (*Return the number of 128 byte blocks in N.*)
SIZ_2K := CONV_SIZE(SIZ_128); (*Return the number of 2k blocks in N.*)
SPCE := CHECK_SPACE(INFILE,SIZ_2K,OFFSET); (*0-63 is block number,
        64 = error.*)

IF SPCE = 64 THEN
BEGIN
  WRITELN('THERE IS NOT ENOUGH SPACE FOR THAT FILE IN MEMORY');
  EXIT;
END;
TOTAL_ENT := MAXENTRY;
EXT := SET_UP_ENTRY(TOTAL_ENT,SPCE,SIZ_128,SIZ_2K); (*Sets up
directory entry, returns the number of extents.*)
MAXENTRY := MAXENTRY + EXT;
UPDATE_DIR(TOTAL_ENT,EXT); (*Update the directory file.*)
CLOSE(INFILE,RESULT);
IF RESULT <> 0 THEN
BEGIN
  WRITELN('CANNOT CLOSE EPDIR.COM');
  EXIT;
END;
NEWEPROM(N,TOTAL_ENT,EXT,SIZ_128,SIZ_2K); (*Creates new EPn.com.*)
CLOSE(N,RESULT);
IF RESULT <> 0 THEN
BEGIN
  WRITELN('CANNOT CLOSE ',NEW_FILE);
  EXIT;
END;
END;

(*---------------------------------------------------------------------------*)
EPROM Management System

PROCEDURE FILE_DELETE;
(* Removes the entry from the File Directory.*)
VAR
    TOTAL_ENT, J, TEM : INTEGER;
    DUP : STRING;
BEGIN
    WRITELN;
    WRITELN;
    WRITELN('DELETE WHICH FILE?');
    READLN(ANS);
    DUP := ANS;
    DELETE(ANS, 1, POS(':', ANS));
    J := POS('.', ANS);
    IF J = 0 THEN
        J := LENGTH(ANS)
    ELSE
        DELETE(ANS, J, 1);
    IF LENGTH(ANS) < 11 THEN
        REPEAT
            INSERT(' ', ANS, J);
        UNTIL LENGTH(ANS) = 11;
    ASSIGN(INFILE, 'EPDIR.COM');
    RESET(INFILE);
IF IORESUL T <> 0 THEN
BEGIN
  WRITELN('THERE IS NO EPDIR.COM ON THIS DISK');
  @HLT;
END;
TOTAL_ENT := READ_DIR(INFILE);
KILL_IT(ANS,DUP,TOTAL_ENT);
TEM := TOTAL_ENT;
TOTAL_ENT := 0;
UPDATE_DIR(TOTAL_ENT,TEM);
WRITELN;
END;
EPROM Management System

(* MAIN LOOP *)

BEGIN
OPEN(INFILE,'EPDIR.COM',RESULT);
IF RESULT = 255 THEN
BEGIN
WRITELN('NO EPDIR.COM ON THIS DISK');
@HLT;
END;
MAXENTRY := READ_DIR(INFILE);
OFFSET := 0;
REPEAT
WRITELN;
WRITELN;
WRITELN(EPROM MANAGEMENT SYSTEM');
WRITELN("=================================================");
WRITELN('Main menu: ');
WRITELN;
WRITELN('Choose one of the following:');
WRITELN('A)dd file');
WRITELN('D)elete file');
WRITELN('M)ap of files');
WRITELN('R)eturn to CP/M');
WRITELN;
READLN(RESPONSE);
EPROM Management System

CASE RESPONSE OF
  'A', 'a' : FILE_ADD(MAXENTRY);
  'D', 'd' : FILE_DELETE;
  'R', 'r' : @HLT;
  'M', 'm' : EPROM_MAP(OFFSET)
ELSE
  WRITELN('Enter ONLY A, D, M or R please')
END;
UNTIL FALSE;
END.
APPENDIX D

SABus LAN Card Circuit Diagrams
SABus LAN Card

Sheet 1 of 2

Designed: QPM

September 85

Drawn: QPM
<table>
<thead>
<tr>
<th>IC#</th>
<th>Type</th>
<th>Power</th>
<th>GND</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>MD2840</td>
<td>48 (5V)</td>
<td>18</td>
</tr>
<tr>
<td></td>
<td></td>
<td>42 (12V)</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>74LS245</td>
<td>20</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>74LS373</td>
<td>20</td>
<td>10</td>
</tr>
<tr>
<td>4</td>
<td>74LS373</td>
<td>20</td>
<td>10</td>
</tr>
<tr>
<td>5</td>
<td>74LS367</td>
<td>16</td>
<td>8</td>
</tr>
<tr>
<td>6</td>
<td>74LS85</td>
<td>16</td>
<td>8</td>
</tr>
<tr>
<td>7</td>
<td>74LS367</td>
<td>16</td>
<td>8</td>
</tr>
<tr>
<td>8</td>
<td>74LS374</td>
<td>20</td>
<td>10</td>
</tr>
<tr>
<td>9</td>
<td>74LS244</td>
<td>20</td>
<td>10</td>
</tr>
<tr>
<td>10</td>
<td>PAL20L10</td>
<td>24</td>
<td>12</td>
</tr>
<tr>
<td>11</td>
<td>PAL20L10</td>
<td>24</td>
<td>12</td>
</tr>
<tr>
<td>12</td>
<td>PAL20L10</td>
<td>24</td>
<td>12</td>
</tr>
<tr>
<td>13</td>
<td>74LS164</td>
<td>14</td>
<td>7</td>
</tr>
<tr>
<td>14</td>
<td>74LS164</td>
<td>14</td>
<td>7</td>
</tr>
<tr>
<td>15</td>
<td>74LS164</td>
<td>14</td>
<td>7</td>
</tr>
<tr>
<td>16</td>
<td>555</td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>17</td>
<td>HD6409</td>
<td>20</td>
<td>10</td>
</tr>
<tr>
<td>18</td>
<td>74LS04</td>
<td>14</td>
<td>7</td>
</tr>
<tr>
<td>19</td>
<td>74LS93</td>
<td>5</td>
<td>10</td>
</tr>
<tr>
<td>20</td>
<td>74LS74</td>
<td>14</td>
<td>7</td>
</tr>
<tr>
<td>21</td>
<td>75176</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>22</td>
<td>74LS74</td>
<td>14</td>
<td>7</td>
</tr>
</tbody>
</table>

**SABus LAN Card**

Sheet 2 of 2 Designed: QPM

September 85 Drawn: QPM
The PAL equations for the SABus LAN card are given below. As before a / means negative logic.

PAL 1

\[ /OC = /WDRDY \times T_1 \times /T_{13} \]

\[ /CNTCLR = T_{23} \times /C_{16M} + /RESET \]

\[ /HOLD = /DRQI \times /T_{13} + /DRQO \times /T_{13} \]

\[ /P = C_{16M} \times /C_{1SEC} \times HOSTCS \times WDRDY + C_{16M} \times /C_{1SEC} \times T_1 \times WDRDY \]

\[ /N = C_{16M} \times \text{INRDY} \times /DRQO + \text{HOLD} \times \text{HOLDA} + C_{16M} \times /\text{INRDY} \times /DRQO \times T_{10} \]

\[ /N = C_{16M} \times \text{INRDY} \times /DRQI + \text{HOLD} \times \text{HOLDA} + C_{16M} \times /\text{INRDY} \times /DRQI \times T_{13} \]

\[ /H = /HOLD \times \text{HOLDA} \]

\[ /\text{COUNTE} = M + N + P \]

PAL 2

\[ /WDRDY = /\text{WRCY} \times /\text{RDCY} \]

\[ /\text{NARCY} = /C_{1SEC} \times WDRDY \times /\text{HOSTCS} + T_0 \times /T_{23} \times /WDRDY \times /C_{1SEC} \]

\[ /\text{OUTRDY} = \text{HOSTCS} \times WDRDY + \text{HOSTCS} \times \text{NARCY} \]

\[ /\text{DOE} = \text{NARCY} \times T_{19} \times /T_{23} \]

\[ /\text{RE} = \text{NARCY} \times T_4 \times /T_{10} \]
PAL Equations for the SABus LAN Card

/WE = NARCY * T15 * /T22

/CS = NARCY * T0 * /T10 + NARCY * T14 * T23 +
     /NARCY * HOSTCS

/ADRE = NARCY * T0 * /T23

/IC2E = /NARCY

PAL 3

/RDCY = /DRQI * T1 * /IC2E

/WRCY = /DRQO * T1 * /IC2E

/MWE = WRCY * T6 * /T10

/MRE = RDCY * T6 * T13

/DACK = T1 * /T12 * WRCY +
       T1 * /T11 * RDCY

/E = /DACK * /OC

/IAB = /RE * /WE + RE * WE

/RCV = WRCY * /OC + HOSTCS * /RE
Timing Diagrams for the WD2840

**WD2840 Control/Status Register Read Timing**

**WD2840 Control/Status Register Write Timing**

**WD2840 DMA Input Timing**

**WD2840 DMA Output Timing**
Timing Diagrams for the WD2840

WD2840 Network Read Timing

WD2840 Network Write Timing
APPENDIX E

The TAC User Routines Program
The Token Access Controller (TAC), WD2840, situated at address 70H to 7FH, is used to control the access to the network. This program enables the user to access the network using three routines, NTWKINIT, which does the network initialisation, SNDMSG, which will transmit a frame pointed to by the DE registers, and RECMSG which receives a frame from the network and writes it to a buffer in memory returning the address in the DE registers. A fourth procedure, STAT, is supplied which returns the status of the network. Here, DE point at a buffer which contains a copy of the 16 registers on the WD2840 as well as a copy of the
SOURCE STATEMENT

31 ; 11 event registers. Lastly, the software enables the higher
32 ; layers to instruct it off the network by using the STNDWN
33 ; entry point.
34 ;
35 ; This software is based on the IEEE 802 standard specifications
36 ; using a Class I LLC layer which only implements an unacknowledged
37 ; connectionless transmission or as it is more commonly known a
38 ; datagram service. The implementation also uses a simple PHY and MAC
39 ; check, which involves verifying byte correctness and, in the case of an
40 ; error the program will inform the user. It will also inform the
41 ; user of the reason for the error in this case.
42 ;
43 ; The addressing is done on a one to one basis throughout the
44 ; PHY, MAC and LLC Layers. This limits the addressing to only
45 ; 63 nodes as defined by the LLC SAPs.
46 ;
47 ; Data is transmitted using the IEEE 802 LLC frame format, although
48 ; the WD2840 does not allow the same PHY frame format. Thus the
49 ; entry to this program looks like a LLC/Network layer interface
50 ; as defined by IEEE.
51 ;
52 ; Responses to both the TEST frame and the XID, exchange
53 ; Identification, control frames are supported by this program.
54 ; To send a control frame the value of the B register is set to
55 ; 0FFH and then the SNDMSG procedure does not add the UI,
56 ; Unnumbered Information, control word but simply transmits the
57 ; data found in the buffer pointed to by the DE register pair.
58 ; The program responds to the TEST and XID frames at the earliest
59 ; possible chance.
60 ;
61 ;================================================================
62 $EJECT
LOC OBJ LINE SOURCE STATEMENT

63 ;================================================================
64 ;
65 ;
66 ;
67 ;================================================================
68 ;
69 ;
70 ;
71 ;
72 ;
73 0000

74 FALSE EQU 0
75 TRUE EQU NOT FALSE
76
77 ;
78 ;
79 ;
80 ;
81 ;
82 000A

83 LF EQU 10 ; Line Feed
84 CR EQU 13 ; Carriage Return
85 CTRLZ EQU 26 ; Control Z
86
87 ;
88 ;
89 ;
90 ;
91 ;
92
<table>
<thead>
<tr>
<th>Line</th>
<th>Source Statement</th>
<th>Address</th>
<th>OBJ</th>
</tr>
</thead>
<tbody>
<tr>
<td>93</td>
<td>OUTMSG</td>
<td>0070</td>
<td>0070</td>
</tr>
<tr>
<td>94</td>
<td>CITEST</td>
<td>0071</td>
<td>0071</td>
</tr>
<tr>
<td>95</td>
<td>CR</td>
<td>0072</td>
<td>0072</td>
</tr>
<tr>
<td>96</td>
<td>DCMP</td>
<td>0073</td>
<td>0073</td>
</tr>
<tr>
<td>97</td>
<td>NETCTRL EQU</td>
<td>0074</td>
<td>0074</td>
</tr>
<tr>
<td>98</td>
<td>; Output a message routine</td>
<td>0075</td>
<td>0075</td>
</tr>
<tr>
<td>99</td>
<td>; Test for console input</td>
<td>0076</td>
<td>0076</td>
</tr>
<tr>
<td>100</td>
<td>; Console input routine</td>
<td>0077</td>
<td>0077</td>
</tr>
<tr>
<td>101</td>
<td>; 16 bit compare</td>
<td>0078</td>
<td>0078</td>
</tr>
<tr>
<td>102</td>
<td>; Controller Equates</td>
<td>0079</td>
<td>0079</td>
</tr>
<tr>
<td>103</td>
<td>;-----------------------------------------------------------------</td>
<td>007A</td>
<td>007A</td>
</tr>
<tr>
<td>104</td>
<td>;-----------------------------------------------------------------</td>
<td>007B</td>
<td>007B</td>
</tr>
<tr>
<td>105</td>
<td>;-----------------------------------------------------------------</td>
<td>007C</td>
<td>007C</td>
</tr>
<tr>
<td>106</td>
<td>;-----------------------------------------------------------------</td>
<td>007D</td>
<td>007D</td>
</tr>
<tr>
<td>107</td>
<td>;-----------------------------------------------------------------</td>
<td>007E</td>
<td>007E</td>
</tr>
<tr>
<td>108</td>
<td>;-----------------------------------------------------------------</td>
<td>007F</td>
<td>007F</td>
</tr>
<tr>
<td>109</td>
<td>;-----------------------------------------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>110</td>
<td>;-----------------------------------------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>111</td>
<td>;-----------------------------------------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>112</td>
<td>;-----------------------------------------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>113</td>
<td>;-----------------------------------------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>114</td>
<td>;-----------------------------------------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>115</td>
<td>;-----------------------------------------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>116</td>
<td>;-----------------------------------------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>117</td>
<td>;-----------------------------------------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>118</td>
<td>;-----------------------------------------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>119</td>
<td>;-----------------------------------------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>120</td>
<td>;-----------------------------------------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>121</td>
<td>;-----------------------------------------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>122</td>
<td>;-----------------------------------------------------------------</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
allows 500us for ACK to return
allows time for transmission
number of cycles skipped on entry
maximum allowable frames/token
the value written to my address
sets up the buffer length to 320 bytes

wait for acknowledge and last frame
wait for acknowledge and last frame
wait for acknowledge and last frame
wait for acknowledge and last frame
wait for acknowledge and last frame

0000 0000 0000 0000 ; Unnumbered frame control word
0000 0000 0000 0000 ; Exchange ID control byte
0000 0000 0000 0000 ; TEST control word
0000 0000 0000 0000 ; The poll bit
0000 0000 0000 0000 ; Inverse poll
<table>
<thead>
<tr>
<th>LOC</th>
<th>OBJ</th>
<th>LINE</th>
<th>SOURCE</th>
<th>STATEMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>00B0</td>
<td>153</td>
<td>CONTWD</td>
<td>EQU 1011000B</td>
<td>; Control word for the USART</td>
</tr>
<tr>
<td>00B1</td>
<td>154</td>
<td>CONT8253</td>
<td>EQU 0BH</td>
<td>; Address of USART control and status</td>
</tr>
<tr>
<td>00B2</td>
<td>155</td>
<td>C28253</td>
<td>EQU 0AH</td>
<td>; Address of the data register</td>
</tr>
<tr>
<td>00B3</td>
<td>156</td>
<td>$EJECT</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Network Initialization

FUNCTION: NTWRKINIT

INPUTS: None

OUTPUTS: CY: set means error

: clear means no error

DESTROYS: A, B, C, D, E, H, L, Flags

DESCRIPTION:

NTWRKINIT sets up the registers in the WD2840, it
will then set up the control block. Once the network
has been checked for duplicate nodes it will attempt
enter the transmission ring. It will return carry
set if there is an error or clear if no error.

SET up the Network parameters (* Station in the DOWN STATE *)

MVI A,THISNODE ; Only this value must be changed on each
STA MYID ; TAC
LXI H,CBLOCK ; Save the ID
LXI D,RXBUFl ; Set up the buffer pointers
MOV M,D ; in the Control Block
INX H
MOV M,E
INX H
LXI D,TXBUFl ; For both receive and transmit buffers
<table>
<thead>
<tr>
<th>LOC</th>
<th>OBJ</th>
<th>LINE</th>
<th>SOURCE STATEMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0021</td>
<td>72</td>
<td>199</td>
<td>MOV M,D</td>
</tr>
<tr>
<td>0022</td>
<td>23</td>
<td>200</td>
<td>INX H</td>
</tr>
<tr>
<td>0023</td>
<td>73</td>
<td>201</td>
<td>MOV M,E</td>
</tr>
<tr>
<td>0024</td>
<td>23</td>
<td>202</td>
<td>INX H</td>
</tr>
<tr>
<td>0025</td>
<td>3604</td>
<td>203</td>
<td>MVI M,BUFLEN     ; Set up the buffer size</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0027</td>
<td>3E10</td>
<td>207</td>
<td>MVI A,RESPONSETIME ; - The length waited for Ack</td>
</tr>
<tr>
<td>0029</td>
<td>D378</td>
<td>208</td>
<td>OUT TA           ; Write to the required addr</td>
</tr>
<tr>
<td>002B</td>
<td>06B0</td>
<td>209</td>
<td>MVI B,TIMEOUT    ; - Network Dead Time</td>
</tr>
<tr>
<td>002D</td>
<td>3A420A C</td>
<td>210</td>
<td>LDA MYID         ; calculated from MA value</td>
</tr>
<tr>
<td>0030</td>
<td>B7</td>
<td>211</td>
<td>ORA A</td>
</tr>
<tr>
<td>0031</td>
<td>1F</td>
<td>212</td>
<td>RAR</td>
</tr>
<tr>
<td>0032</td>
<td>80</td>
<td>213</td>
<td>ADD B</td>
</tr>
<tr>
<td>0033</td>
<td>D379</td>
<td>214</td>
<td>OUT TD</td>
</tr>
<tr>
<td>0035</td>
<td>212F05 C</td>
<td>215</td>
<td>LXI H,CBLOCK     ; Fetch the Control Block addr</td>
</tr>
<tr>
<td>0038</td>
<td>7D</td>
<td>216</td>
<td>MOV A,L          ; - Output the lower address</td>
</tr>
<tr>
<td>0039</td>
<td>D37B</td>
<td>217</td>
<td>OUT CBPL         ; - Then the higher address</td>
</tr>
<tr>
<td>003B</td>
<td>7C</td>
<td>218</td>
<td>MOV A,H</td>
</tr>
<tr>
<td>003C</td>
<td>D37A</td>
<td>219</td>
<td>OUT CBPH</td>
</tr>
<tr>
<td>003E</td>
<td>3A420A C</td>
<td>220</td>
<td>LDA MYID         ; - Set up the Next Address</td>
</tr>
<tr>
<td>0041</td>
<td>3C</td>
<td>221</td>
<td>INR A</td>
</tr>
<tr>
<td>0042</td>
<td>D37C</td>
<td>222</td>
<td>OUT NAR</td>
</tr>
<tr>
<td>0044</td>
<td>3E00</td>
<td>223</td>
<td>MVI A,CYCLESKIP  ; - Miss the required cycles</td>
</tr>
<tr>
<td>0046</td>
<td>D37D</td>
<td>224</td>
<td>OUT AHOLT</td>
</tr>
<tr>
<td>0048</td>
<td>3E01</td>
<td>225</td>
<td>MVI A,MAXFRAMES  ; - Store max frames/token</td>
</tr>
<tr>
<td>004A</td>
<td>D37E</td>
<td>226</td>
<td>OUT TSLT</td>
</tr>
<tr>
<td>004C</td>
<td>3A420A C</td>
<td>227</td>
<td>LDA MYID         ; - My address</td>
</tr>
<tr>
<td>004F</td>
<td>D37F</td>
<td>228</td>
<td>OUT MA</td>
</tr>
</tbody>
</table>
LOC OBJ LINE SOURCE STATEMENT

229
230 ; Dummy Start, checking for Network dead

0051 3E00 232 MVI A,00 ; Set the two control regs to zero
0053 D371 233 OUT CR1 ;
0055 D370 234 OUT CR0 ;
235 STCHG: ;
0057 DB75 236 IN SR2 ; Wait for the State Change
0059 E602 237 ANI 02H ; - indicates ready
005B E3 238 XTHL ; Create a delay for the WD
005C E3 239 XTHL ; to update its registers
005D C25700 C 240 JNZ STCHG ;
0060 0EFF 241 ;
0062 DB73 242 MVI C,0FFH ; Counter for Software loop
0064 E601 243 LOOPD: ;
0066 C2A000 C 244 IN IR0 ; Is the network dead?
0069 E3 245 ANI 01H ; Set if time TD passes
006A E3 246 JNZ DEAD ;
006B 0D 247 XTHL ; Delay for WD reg update
006C C26200 C 248 XTHL ;
006D E3 249 DCR C ;
006E C26200 C 250 JNZ LOOPD ; If C = 0 then the Network is not dead
006F 3E01 251 ;
0071 D371 252 NONDEAD: ;
0073 3E10 253 MVI A,01H ; - Take NAR
0075 D370 254 OUT CR1 ;
255 MVI A,10H ; - Enable Token Interrupts
256 OUT CR0 ;
257 ; Watch the ITOK for addr dup (* Station in DUPLICATE ADDRESS CHECK STATE *)
<table>
<thead>
<tr>
<th>LOC</th>
<th>OBJ</th>
<th>LINE</th>
<th>SOURCE STATEMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0077</td>
<td>015046</td>
<td>260</td>
<td>LXI B,18000; This will keep looking for 0.5 sec</td>
</tr>
<tr>
<td>007A</td>
<td>3E00</td>
<td>261</td>
<td>MVI A,0; Reset counter</td>
</tr>
<tr>
<td>007C</td>
<td>32500A</td>
<td>262</td>
<td>STA ADDUP;</td>
</tr>
<tr>
<td></td>
<td>C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>007F</td>
<td>DB73</td>
<td>263</td>
<td>DIAG1:</td>
</tr>
<tr>
<td>0081</td>
<td>6E04</td>
<td>264</td>
<td>IN 73H; Read interrupt register</td>
</tr>
<tr>
<td>0083</td>
<td>C29100</td>
<td>265</td>
<td>ANI 04H; ITOK bit set?</td>
</tr>
<tr>
<td>0086</td>
<td>E3</td>
<td>266</td>
<td>JNZ FDIAG;</td>
</tr>
<tr>
<td>0087</td>
<td>E3</td>
<td>267</td>
<td>STA ADDUP;</td>
</tr>
<tr>
<td>0088</td>
<td>0B</td>
<td>268</td>
<td>STA ADDUP;</td>
</tr>
<tr>
<td>0089</td>
<td>78</td>
<td>269</td>
<td>DCK B; Time up?</td>
</tr>
<tr>
<td>008A</td>
<td>B1</td>
<td>270</td>
<td>MOV A,B;</td>
</tr>
<tr>
<td>008B</td>
<td>C27F00</td>
<td>271</td>
<td>ORA C;</td>
</tr>
<tr>
<td>008E</td>
<td>C3A000</td>
<td>272</td>
<td>JNZ DIAG1;</td>
</tr>
<tr>
<td></td>
<td>C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0091</td>
<td>3A500A</td>
<td>273</td>
<td>JMP DEAD;</td>
</tr>
<tr>
<td>0094</td>
<td>3C</td>
<td>274</td>
<td>FDIAG:</td>
</tr>
<tr>
<td>0095</td>
<td>FE02</td>
<td>275</td>
<td>LDA ADDUP;</td>
</tr>
<tr>
<td>0097</td>
<td>CADD00</td>
<td>276</td>
<td>INR A;</td>
</tr>
<tr>
<td>0099</td>
<td>32500A</td>
<td>277</td>
<td>CPI 2;</td>
</tr>
<tr>
<td>009A</td>
<td>32500A</td>
<td>278</td>
<td>JZ FAILSDIAG;</td>
</tr>
<tr>
<td>009D</td>
<td>C37F00</td>
<td>279</td>
<td>STA ADDUP;</td>
</tr>
<tr>
<td></td>
<td>C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00A0</td>
<td>DB71</td>
<td>280</td>
<td>JMP DIAG1;</td>
</tr>
<tr>
<td>00A2</td>
<td>F628</td>
<td>281</td>
<td></td>
</tr>
<tr>
<td>00A4</td>
<td>D371</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00A6</td>
<td>3E50</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

; Get into the logical loop

283

284 DEAD:

285 IN CR1; Get into the logical loop

286 ORI 28H;

287 OUT CR1;

288 MVI A,50H; RXEN set
LOC OBJ
LINE SOURCE STATEMENT

289 OUT 290 CR0

00A8 0370 289 OUT CR0

STATE clear confirms change of IRING set means token passed

00AA AC03 291 IN SI

00AC E613 292 ANI CR1

00AE FE01 293 PUSH PSW

00B0 0101 294 CALL OLAY

00B4 F101 295 POP PSW

00B5 C000 296 JNZ READY

Enable the transmission of the frames

00B8 3E42 297 MVI A,042H

Enable transmission, disable ITOK interrupts

00BA 0370 298 OUT CR0

Set up buffer pointers

00BC 3E02 299 MVI A,2

Do the required initialisation of memory

00BE 3240 300 IN SI

Set up Buffer-to-use pointer

00BF C101 301 PULL CR0

Reset receive buffer FSBs

00C9 3209 302 STA ?RXBUF

Set transmit buffer FSBs to DONE

00CC 3210 303 STA RXBUF1+2

Station in UP STATE

00CE 3211 304 STA RXBUF2+2

Finish (*)

00D1 3201 305 STA TXBUF1+2

Inform the user

00D4 21CB 306 STA TXBUF2+2

00D7 3212 307 STA TXBUF1+2

00D8 3213 308 STA TXBUF2+2

00D9 3214 309 STA TXBUF1+2

00DA 3215 310 STA TXBUF2+2

00DB 3216 311 STA TXBUF1+2

00DC 3217 312 STA TXBUF2+2

00DD 3218 313 STA TXBUF1+2

00DE 3219 314 STA TXBUF2+2

00DF 321A 315 STA TXBUF1+2

00E0 321B 316 STA TXBUF2+2

00E1 321C 317 STA TXBUF1+2

00E2 321D 318 STA TXBUF2+2

00E3 F9 319 STA H,READY

00E4 319 31A STA H,READY

00E5 319 31B STA H,READY

00E6 319 31C STA H,READY

00E7 319 31D STA H,READY

00E8 319 31E STA H,READY

00E9 319 31F STA H,READY
LOC | OBJ   | LINE | SOURCE STATEMENT |
-----|-------|------|------------------|
00D7 | CD3DF0| 319  | CALL OUTMSG      |
<pre><code>  |       | 320  |                  |
  |       | 321  | STC              |
  |       | 322  | CMC              |
  |       | 323  | RET              |
  |       | 324  |                  |
  |       | 325  | ; Exit here if error  |
  |       | 326  | (* Station in DOWN STATE *) |
  |       | 327  | FAILSDIAG:       |
  |       | 328  | LXI H,FAILMSG    |
  |       | 329  | CALL OUTMSG      |
  |       | 330  |                  |
  |       | 331  |                  |
  |       | 332  |                  |
  |       | 333  | $EJECT           |
</code></pre>
<table>
<thead>
<tr>
<th>LOC</th>
<th>OBJ</th>
<th>LINE</th>
<th>SOURCE STATEMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>334</td>
<td></td>
<td>;</td>
<td>;</td>
</tr>
<tr>
<td>335</td>
<td></td>
<td>;</td>
<td>Brings the station off-line</td>
</tr>
<tr>
<td>336</td>
<td></td>
<td>;</td>
<td>;</td>
</tr>
<tr>
<td>337</td>
<td></td>
<td>;</td>
<td>FUNCTION: STNDWN</td>
</tr>
<tr>
<td>338</td>
<td></td>
<td>;</td>
<td>INPUTS : None</td>
</tr>
<tr>
<td>339</td>
<td></td>
<td>;</td>
<td>OUTPUTS : A = 0 the isolation confirmation is pending</td>
</tr>
<tr>
<td>340</td>
<td></td>
<td>;</td>
<td>A = 2 the isolation is complete</td>
</tr>
<tr>
<td>341</td>
<td></td>
<td>;</td>
<td>Carry Bit set</td>
</tr>
<tr>
<td>342</td>
<td></td>
<td>;</td>
<td>DESTROYS: A, B, Flags</td>
</tr>
<tr>
<td>343</td>
<td></td>
<td>;</td>
<td>DESCRIPTION:</td>
</tr>
<tr>
<td>344</td>
<td></td>
<td>;</td>
<td>Sets the WD2840 into off-line state and waits for</td>
</tr>
<tr>
<td>345</td>
<td></td>
<td>;</td>
<td>200 ms for confirmation after which it will set</td>
</tr>
<tr>
<td>346</td>
<td></td>
<td>;</td>
<td>the accumulator to indicate confirmation pending</td>
</tr>
<tr>
<td>347</td>
<td></td>
<td>;</td>
<td>and return. If the confirmation occurs in the</td>
</tr>
<tr>
<td>348</td>
<td></td>
<td>;</td>
<td>200 ms provided then the accumulator returns with</td>
</tr>
<tr>
<td>349</td>
<td></td>
<td>;</td>
<td>a value of 2. The carry bit is set to enable</td>
</tr>
<tr>
<td>350</td>
<td></td>
<td>;</td>
<td>compatibility with the CP/NET status returns.</td>
</tr>
<tr>
<td>351</td>
<td></td>
<td>;</td>
<td>;</td>
</tr>
<tr>
<td>352</td>
<td></td>
<td>;</td>
<td>;</td>
</tr>
<tr>
<td>353</td>
<td>3E01</td>
<td>;</td>
<td>STNDWN:</td>
</tr>
<tr>
<td>354</td>
<td></td>
<td>;</td>
<td>MVI A, 01H</td>
</tr>
<tr>
<td>355</td>
<td></td>
<td>;</td>
<td>OUT CR0</td>
</tr>
<tr>
<td>356</td>
<td></td>
<td>;</td>
<td>MVI B, 20</td>
</tr>
<tr>
<td>357</td>
<td></td>
<td>;</td>
<td>WAITONSTAT:</td>
</tr>
<tr>
<td>358</td>
<td></td>
<td>;</td>
<td>CALL DLAY</td>
</tr>
<tr>
<td>359</td>
<td></td>
<td>;</td>
<td>IN SR2</td>
</tr>
<tr>
<td>360</td>
<td></td>
<td>;</td>
<td>ANI 02H</td>
</tr>
<tr>
<td>361</td>
<td></td>
<td>;</td>
<td>JNZ DWNITIS</td>
</tr>
<tr>
<td>362</td>
<td></td>
<td>;</td>
<td>DCR B</td>
</tr>
<tr>
<td>363</td>
<td></td>
<td>;</td>
<td>JNZ WAITONSTAT</td>
</tr>
</tbody>
</table>

FUNCTION: STNDWN

Brings the station off-line.

INPUTS:
- None

OUTPUTS:
- A = 0: the isolation confirmation is pending
- A = 2: the isolation is complete
- Carry Bit set

DESTROYS:
- A, B, Flags

DESCRIPTION:

Sets the WD2840 into off-line state and waits for 200 ms for confirmation after which it will set the accumulator to indicate confirmation pending and return. If the confirmation occurs in the 200 ms provided then the accumulator returns with a value of 2. The carry bit is set to enable compatibility with the CP/NET status returns.
<table>
<thead>
<tr>
<th>LOC</th>
<th>OBJ</th>
<th>LINE</th>
<th>SOURCE</th>
<th>STATEMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>00F7</td>
<td>3E00</td>
<td>364</td>
<td>MVI</td>
<td>A,0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>; Indicate pending state</td>
</tr>
<tr>
<td>00F9</td>
<td>21DD03</td>
<td>365</td>
<td>DWNITIS:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>C</td>
<td></td>
<td></td>
<td>;</td>
</tr>
<tr>
<td>00FC</td>
<td>CD3DF0</td>
<td>366</td>
<td>LXI</td>
<td>H,OUTITIS</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>; Inform user</td>
</tr>
<tr>
<td>00FF</td>
<td>37</td>
<td>367</td>
<td>CALL</td>
<td>OUTMSG</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>;</td>
</tr>
<tr>
<td>0100</td>
<td>C9</td>
<td>368</td>
<td>STC</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>; Set the carry as a flag</td>
</tr>
<tr>
<td></td>
<td></td>
<td>369</td>
<td>RET</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>370</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>371</td>
<td>$EJECT</td>
<td></td>
</tr>
</tbody>
</table>
## FUNCTION: DLAY

**INPUTS**: None  
**OUTPUTS**: None  
**DESTROYS**: None  
**DESCRIPTION**: Delays the CPU for 10 ms.

### Delays the CPU for 10 ms

```assembly
; Delays for 10 ms
; Save present status

0101 C5 384 PUSH B
0102 F5 385 PUSH PSW
0103 01FD04 386 LXI B,1277
0106 0B 387 DLL10:
0107 78 388 DCX B
0108 B1 389 MOV A,B
0109 C20601 C 390 ORA C
010C F1 391 JNZ DLL10
010D C1 392 ; Loop till time up
010E C9 393 POP PSW
010F $EJECT 394 POP B
0110  
```

Delay for 10 ms:

- Save present status
- Decrement counter
- Loop till time up
- Restore status
FUNCTION: SNDMSG

INPUTS: DE pointing at a buffer containing data
Buffer - Length (2 Bytes), DA, [IEEE control byte,]
Data bytes, (x length)
Flags: A = 0 means send and don't wait for ACK
B = 0FFH means the frame is a control frame

OUTPUTS: A = 0 then successful
A = Non-zero then unsuccessful and DE points to the
beginning of the section of the buffer which
failed. The error codes are:
A = 1 - Insufficient buffers at the receiver
A = 2 - Receiver not enabled
A = 3 - Receiver over-run
A = 4 - Frame exceeded 16 receiver buffers
B = 0 means send and wait for ACK after frame TX
B = 0FFH means a control frame

DESTROYS: A, B, C, D, E, H, L, Flags

DESCRIPTION: Takes the message from the pointed to by the DE
registers and sends it across the network. The max
frame is 311 bytes, and so if the frame is longer then SNDMSG will break the contents of the buffer into 311 byte blocks and send them. If any frame is unsuccessful on two attempts then the A register is set to 1 and the routine returns with the DE register pair pointing to the start of the unsuccessful block. If the total transmission is successful then A is set to 0 and the routine returns. The value of DE is irrelevant in the second case. A IEEE control word is added automatically if the value of the B register is 0 on entry, if it is 0FFH then the control word is expected in the buffer.

<table>
<thead>
<tr>
<th>LOC</th>
<th>OBJ</th>
<th>LINE</th>
<th>SOURCE STATEMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>429</td>
<td>;</td>
<td></td>
<td>frame is 311 bytes, and so if the frame is longer</td>
</tr>
<tr>
<td>430</td>
<td>;</td>
<td></td>
<td>then SNDMSG will break the contents of the buffer</td>
</tr>
<tr>
<td>431</td>
<td>;</td>
<td></td>
<td>into 311 byte blocks and send them. If any frame</td>
</tr>
<tr>
<td>432</td>
<td>;</td>
<td></td>
<td>is unsuccessful on two attempts then the A register</td>
</tr>
<tr>
<td>433</td>
<td>;</td>
<td></td>
<td>is set to 1 and the routine returns with the DE</td>
</tr>
<tr>
<td>434</td>
<td>;</td>
<td></td>
<td>register pair pointing to the start of the</td>
</tr>
<tr>
<td>435</td>
<td>;</td>
<td></td>
<td>unsuccessful block. If the total transmission is</td>
</tr>
<tr>
<td>436</td>
<td>;</td>
<td></td>
<td>successful then A is set to 0 and the routine returns.</td>
</tr>
<tr>
<td>437</td>
<td>;</td>
<td></td>
<td>The value of DE is irrelevant in the second case.</td>
</tr>
<tr>
<td>438</td>
<td>;</td>
<td></td>
<td>A IEEE control word is added automatically if the</td>
</tr>
<tr>
<td>439</td>
<td>;</td>
<td></td>
<td>value of the B register is 0 on entry, if it is 0FFH</td>
</tr>
<tr>
<td>440</td>
<td>;</td>
<td></td>
<td>then the control word is expected in the buffer.</td>
</tr>
<tr>
<td>441</td>
<td>;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>442</td>
<td>;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>443</td>
<td>;</td>
<td></td>
<td>SNDMSG:</td>
</tr>
<tr>
<td>444</td>
<td>;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>445</td>
<td>;</td>
<td></td>
<td>Read the flags first</td>
</tr>
<tr>
<td>446</td>
<td>;</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0107</td>
<td>3C</td>
<td>447</td>
<td>INR A ; Send with ACK?</td>
</tr>
<tr>
<td>0110</td>
<td>CA1801</td>
<td>448</td>
<td>JZ ACKIT ; Yes</td>
</tr>
<tr>
<td>0113</td>
<td>3E00</td>
<td>449</td>
<td>MVI A, FALSE ; No</td>
</tr>
<tr>
<td>0115</td>
<td>C31A01</td>
<td>450</td>
<td>JMP SML1</td>
</tr>
<tr>
<td>0118</td>
<td>3EFF</td>
<td>451</td>
<td>ACKIT:</td>
</tr>
<tr>
<td>0119</td>
<td>3EFF</td>
<td>452</td>
<td>MVI A, TRUE</td>
</tr>
<tr>
<td>011A</td>
<td>32470A</td>
<td>453</td>
<td>SML1:</td>
</tr>
<tr>
<td>011D</td>
<td>78</td>
<td>454</td>
<td>STA SNDWITHACK ; Save in the relevant flag</td>
</tr>
<tr>
<td>011E</td>
<td>3C</td>
<td>455</td>
<td></td>
</tr>
<tr>
<td>011F</td>
<td>CA2701</td>
<td>456</td>
<td>MOV A, B ; Read the control frame flag</td>
</tr>
<tr>
<td></td>
<td></td>
<td>457</td>
<td>INR A ; Control Frame?</td>
</tr>
<tr>
<td></td>
<td></td>
<td>458</td>
<td>JZ CTLFRM ; Yes</td>
</tr>
</tbody>
</table>
; No
; Read Length byte
; and save it
; Read Destination Byte
; and save it
; Save the pointer to the first byte
; of the new frame
; Read the Buffer to be used
; ?TXBUF contains value of last buffer used—use the other one
LOC      OBJ  LINE  SOURCE STATEMENT
014A  C25801  C  489  JNZ  USED2
014D  21FF08  C  490  LXI  H,TXBUF2
0150  3E02    491  MVI  A,2
0152  32400A  C  492  STA  ?TXBUF   ; And save the status
0155  C36001  C  493  JMP  FILL
0158  21BF07  C  494  LXI  H,TXBUF1
015B  3E01    496  MVI  A,1
015D  32400A  C  497  STA  ?TXBUF
015F  21BF07  C  498  LXI  H,TXBUF1
0160  3600    502  MVI  M,0   ; Clear the links
0162  23      503  INX  H
0163  3600    504  MVI  M,0
0165  23      505  INX  H
0166  3600    506  MVI  M,0   ; Clear FSB
0168  23      507  INX  H
0169  3A470A  C  511  LDA  SNDWITHACK   ; ACK or not?
016C  FE00    512  CPI  FALSE   ; No
016E  CA7E01  C  513  JZ   F01
0171  3A4B0A  C  514  LDA  DEST   ; If a broadcast frame dont wait for ack
0174  FEFF    515  CPI  0FFH   ; 0FFH = Broadcast Frame
0176  CA7E01  C  516  JZ   F01
0179  36C0    517  MVI  M,FCBWAIT   ; Set FCB to wait for response
017B  C38001  C  518  JMP  F02
LOC  OBJ  LINE  SOURCE STATEMENT

017E  3600  519  F01:  MVI  M,FCBNOWT  ; Set the FCB to no waiting for ACK
      520
      521  F02:  
      0180  23  522  INX  H  
      523  524 ; Calculate number of buffers to be used
      525
      0181  D5  526  PUSH  D  
      0182  E5  527  PUSH  H  
      0183  2A450A  C  528  LHLH  LEN  ; If LEN < Buffer Then
      0184  113601  529  LXIl  D,BYTECOUNT-10  ; Only one buffer
      0185  CD43F0  530  CALL  DCMP  ; Retrieve LENGTH field addr
      0186  DAAF01  C  531  JC  LTBUF  ; First the MSB then the LSB
      0187  E1  532  POP  H  
      0188  113F01  533  LXIl  D,BYTECOUNT-1  ; Write max length to it
      0189  72  534  MOV  M,D  
      018A  23  535  INX  H  
      018B  73  536  MOV  M,E  
      018C  23  537  INX  H  
      018D  23  538  PUSH  H  
      018E  2A450A  C  539  LHLH  LEN  ; Adjust the length buffer
      018F  11CAFE  540  LXIl  D,-(BYTECOUNT-10)  ;
      0190  19  541  DAD  D  
      0191  22450A  C  542  SHEL  LEN  
      0192  E1  543  POP  H  
      0193  D1  544  POP  D  
      0194  013601  545  LXIl  B,BYTECOUNT-10  ; Set counter
      0195  3EFP  546  MVI  A,TRUE  ; Indicate frame > Buffer
      0196  32440A  C  547  STA  LONG  
      0197  C3C701  C  548  JMP  FILDEST  ;
<table>
<thead>
<tr>
<th>LOC</th>
<th>OBJ</th>
<th>LINE</th>
<th>SOURCE STATEMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>01AF</td>
<td>44</td>
<td>549</td>
<td>LTBUF:</td>
</tr>
<tr>
<td>01B0</td>
<td>4D</td>
<td>550</td>
<td>MOV B,H</td>
</tr>
<tr>
<td>01B1</td>
<td>110900</td>
<td>551</td>
<td>MOV C,L</td>
</tr>
<tr>
<td>01B4</td>
<td>19</td>
<td>552</td>
<td>LXI D,9</td>
</tr>
<tr>
<td>01B5</td>
<td>EB</td>
<td>553</td>
<td>DAD D</td>
</tr>
<tr>
<td>01B6</td>
<td>210000</td>
<td>554</td>
<td>XCHG</td>
</tr>
<tr>
<td>01B9</td>
<td>22450A</td>
<td>555</td>
<td>LXI H,0</td>
</tr>
<tr>
<td>01BC</td>
<td>E1</td>
<td>556</td>
<td>MOV M,D</td>
</tr>
<tr>
<td>01BD</td>
<td>72</td>
<td>557</td>
<td>INX H</td>
</tr>
<tr>
<td>01BE</td>
<td>23</td>
<td>558</td>
<td>MOV M,E</td>
</tr>
<tr>
<td>01BF</td>
<td>73</td>
<td>559</td>
<td>INX H</td>
</tr>
<tr>
<td>01C0</td>
<td>23</td>
<td>560</td>
<td>MVIA, FALSE</td>
</tr>
<tr>
<td>01C3</td>
<td>3244A</td>
<td>561</td>
<td>STA LONG</td>
</tr>
<tr>
<td>01C6</td>
<td>D1</td>
<td>562</td>
<td>POP D</td>
</tr>
<tr>
<td></td>
<td></td>
<td>563</td>
<td>FILDEST:</td>
</tr>
<tr>
<td>01C7</td>
<td>3A4B0A</td>
<td>570</td>
<td>LDA DEST</td>
</tr>
<tr>
<td>01CA</td>
<td>77</td>
<td>571</td>
<td>MOV M,A</td>
</tr>
<tr>
<td>01CB</td>
<td>3A420A</td>
<td>572</td>
<td>LDA MYID</td>
</tr>
<tr>
<td>01CE</td>
<td>23</td>
<td>573</td>
<td>INX H</td>
</tr>
<tr>
<td>01CF</td>
<td>77</td>
<td>574</td>
<td>MOV M,A</td>
</tr>
<tr>
<td>01D0</td>
<td>EB</td>
<td>575</td>
<td>XCHG</td>
</tr>
<tr>
<td>01D1</td>
<td>2A4C0A</td>
<td>576</td>
<td>LHLD LASTBYTE</td>
</tr>
<tr>
<td>01D4</td>
<td>224E0A</td>
<td>577</td>
<td>SHLD BEGBUF</td>
</tr>
</tbody>
</table>

; Frame Less than one buffer long
; Set up the counter
; Adjust LEN for LENGTH
; Reset LEN
; Store in LENGTH
; Fill in the rest of the buffer
; Read the Destination Address
; Read the Source Address
; Read addr of last byte
; Save it for the error return
<table>
<thead>
<tr>
<th>LOC</th>
<th>OBJ</th>
<th>LINE</th>
<th>SOURCE STATEMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>01D7</td>
<td>EB</td>
<td>579</td>
<td>XCHG</td>
</tr>
<tr>
<td>01D8</td>
<td>23</td>
<td>580</td>
<td></td>
</tr>
<tr>
<td>01D9</td>
<td>3A480A</td>
<td>581</td>
<td>INX H</td>
</tr>
<tr>
<td>01DC</td>
<td>FEFF</td>
<td>582</td>
<td>LDA CTLFR</td>
</tr>
<tr>
<td>01DE</td>
<td>CAE401</td>
<td>583</td>
<td>CPI TRUE</td>
</tr>
<tr>
<td>01E1</td>
<td>3603</td>
<td>584</td>
<td>JZ COPYIT</td>
</tr>
<tr>
<td>01E3</td>
<td>23</td>
<td>585</td>
<td>MVI M,UI</td>
</tr>
<tr>
<td></td>
<td></td>
<td>586</td>
<td>INX H</td>
</tr>
<tr>
<td>01E4</td>
<td>CDA603</td>
<td>587</td>
<td>COPYIT:</td>
</tr>
<tr>
<td>01E7</td>
<td>EB</td>
<td>588</td>
<td>CALL COPY</td>
</tr>
<tr>
<td>01E8</td>
<td>224C0A</td>
<td>589</td>
<td>XCHG</td>
</tr>
<tr>
<td>01EB</td>
<td>21BF07</td>
<td>590</td>
<td>SHLD LASTBYTE</td>
</tr>
<tr>
<td>01EE</td>
<td>11FF08</td>
<td>591</td>
<td></td>
</tr>
<tr>
<td>01F1</td>
<td>3A400A</td>
<td>592</td>
<td>LXI H,TXBUF1</td>
</tr>
<tr>
<td>01F4</td>
<td>FE02</td>
<td>593</td>
<td>LXI D,TXBUF2</td>
</tr>
<tr>
<td>01F6</td>
<td>CAFA01</td>
<td>594</td>
<td>LDA ?TXBUF</td>
</tr>
<tr>
<td>01F9</td>
<td>EB</td>
<td>595</td>
<td>CPI 2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>596</td>
<td>JZ USED1</td>
</tr>
<tr>
<td>01FA</td>
<td>23</td>
<td>597</td>
<td>XCHG</td>
</tr>
<tr>
<td>01FB</td>
<td>73</td>
<td>598</td>
<td>USED1:</td>
</tr>
<tr>
<td>01FC</td>
<td>2B</td>
<td>599</td>
<td>INX H</td>
</tr>
<tr>
<td>01FD</td>
<td>72</td>
<td>600</td>
<td>MOV M,E</td>
</tr>
<tr>
<td></td>
<td></td>
<td>601</td>
<td>DCX H</td>
</tr>
<tr>
<td></td>
<td></td>
<td>602</td>
<td>MOV M,D</td>
</tr>
<tr>
<td>603</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>604</td>
<td></td>
<td></td>
<td>; Allow to transmit</td>
</tr>
<tr>
<td>605</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>01FE</td>
<td>DB70</td>
<td>606</td>
<td>IN CR0</td>
</tr>
<tr>
<td>0200</td>
<td>F6C0</td>
<td>607</td>
<td>ORI 11000000B</td>
</tr>
<tr>
<td>0202</td>
<td>D370</td>
<td>608</td>
<td>OUT CR0</td>
</tr>
</tbody>
</table>

; A control frame?
; Supply the control byte
; Adjust the pointer
; Copy (BC) bytes from (DE) to (HL)
; Save the addr of last byte
; Find the last buffer address
; Link up to the previous buffer
; Least significant byte first
; Read present value
; Enable transmission
<table>
<thead>
<tr>
<th>LOC</th>
<th>OBJ</th>
<th>LINE</th>
<th>SOURCE</th>
<th>STATEMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>609</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>610</td>
<td>EB</td>
<td>0204</td>
<td>XCHG</td>
<td>; Wait for transmission</td>
</tr>
<tr>
<td>611</td>
<td></td>
<td>0205</td>
<td>INX H</td>
<td>;</td>
</tr>
<tr>
<td>612</td>
<td></td>
<td>0206</td>
<td>INX H</td>
<td>; Point at the FSB</td>
</tr>
<tr>
<td>613</td>
<td></td>
<td>0207</td>
<td>MOV A,M</td>
<td>; Read FSB</td>
</tr>
<tr>
<td>614</td>
<td></td>
<td>0208</td>
<td>MOV B,A</td>
<td>;</td>
</tr>
<tr>
<td>615</td>
<td></td>
<td>0209</td>
<td>ANI 1000000B</td>
<td>; DONE bit set?</td>
</tr>
<tr>
<td>616</td>
<td></td>
<td>020B</td>
<td>JNZ ENDING</td>
<td>;</td>
</tr>
<tr>
<td>617</td>
<td></td>
<td>020E</td>
<td>CALL DLAY</td>
<td>; Wait for WD to update memory</td>
</tr>
<tr>
<td>618</td>
<td></td>
<td>0211</td>
<td>JMP ?SENT</td>
<td>;</td>
</tr>
<tr>
<td>619</td>
<td></td>
<td>0214</td>
<td>MOV A,B</td>
<td>;</td>
</tr>
<tr>
<td>620</td>
<td></td>
<td>0215</td>
<td>ANI Ø7H</td>
<td>; Was it successful</td>
</tr>
<tr>
<td>621</td>
<td></td>
<td>0217</td>
<td>JNZ END2</td>
<td>; - No</td>
</tr>
<tr>
<td>622</td>
<td></td>
<td>021A</td>
<td>LDA LONG</td>
<td>; If LONG = TRUE</td>
</tr>
<tr>
<td>623</td>
<td></td>
<td>021D</td>
<td>CPI TRUE</td>
<td>; Then Loop</td>
</tr>
<tr>
<td>624</td>
<td></td>
<td>021F</td>
<td>JNZ END2</td>
<td>;</td>
</tr>
<tr>
<td>625</td>
<td></td>
<td>0222</td>
<td>CALL DLAY</td>
<td>;</td>
</tr>
<tr>
<td>626</td>
<td></td>
<td>0225</td>
<td>JMP SM1</td>
<td>;</td>
</tr>
<tr>
<td>627</td>
<td></td>
<td>0228</td>
<td>LHLDB BEGBUF</td>
<td>; Find beginning of last block sent</td>
</tr>
<tr>
<td>628</td>
<td></td>
<td>022B</td>
<td>XCHG</td>
<td>;</td>
</tr>
<tr>
<td>629</td>
<td></td>
<td>022C</td>
<td>LDA PRESSTAT</td>
<td>; Restore CRØ</td>
</tr>
<tr>
<td>630</td>
<td>EB</td>
<td>022F</td>
<td>OUT CRØ</td>
<td>;</td>
</tr>
<tr>
<td>LOC</td>
<td>OBJ</td>
<td>LINE</td>
<td>SOURCE</td>
<td>STATEMENT</td>
</tr>
<tr>
<td>-----</td>
<td>-----</td>
<td>------</td>
<td>--------</td>
<td>-----------</td>
</tr>
<tr>
<td>0231</td>
<td>78</td>
<td>639</td>
<td>MOV</td>
<td>A,B ; Clear top nibble</td>
</tr>
<tr>
<td>0232</td>
<td>E60F</td>
<td>640</td>
<td>ANI</td>
<td>0FH ; 8H &lt;&gt; Error</td>
</tr>
<tr>
<td>0234</td>
<td>FE08</td>
<td>641</td>
<td>CPI</td>
<td>08H ;</td>
</tr>
<tr>
<td>0236</td>
<td>C23B02</td>
<td>C</td>
<td>642</td>
<td>JNZ</td>
</tr>
<tr>
<td>0239</td>
<td>3E00</td>
<td>643</td>
<td>MVI</td>
<td>A,0 ; If A = 08H then set to 0</td>
</tr>
<tr>
<td>023B</td>
<td>C9</td>
<td>644</td>
<td>END3:</td>
<td></td>
</tr>
<tr>
<td>645</td>
<td>RET</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>646</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>647</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>648</td>
<td>$EJECT</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
FUNCTION : RECMSG
INPUTS : None
OUTPUTS : DE - buffer

DESCRIPTION: The RECMSG procedure will take the frame received from network and write it to a buffer in the following format: Len (2 BYTES), SA, data bytes (x Len)
The DE registers are then set up to point at the buffer.

------------------------------------------------------------------------
RECMG:
------------------------------------------------------------------------

023C  DB70 649  IN  CR0  ; Read present status
023E  F660 650  ORI  01100000B  ; Enable receive
0240  D370 651  OUT  CR0  ;
0242  3E2C 652  MVI  A,2CH  ; Set up CR1
0244  D371 653  OUT  CR1  ;
0246  3A3F0A C 654  LDA  ?RXBUF  ; Find out which buffer was used
0249  FE01 655  CPI  1  ; last time and use the other
024B  C25C02 C 656  JNZ  RX2  ;
024E  217F06 C 657  LXI  H,RXBUF2  ;
0251  3E02 658  MVI  A,2  ;
0253  32410A C 659  STA  N?RXBUF  ; Save the new buffer in a temporary
0256  113F05 C 660  LXI  D,RXBUF1  ; store
LOC | OBJ | LINE | SOURCE | STATEMENT
---|-----|------|--------|--------
0259 C36702 C 679 | JMP | REC |
0250 RX2: |
025C 213E05 C 681 | LXI | H,RXBUF1 |
025F 3E01 C 682 | MVI | A,1 |
0261 32410A C 683 | STA | N?RXBUF |
0264 117F06 C 684 | LXI | D,RXBUF2 |
0265 |
0266 ; The buffer is now known, see if any data has been received |
0267 23 |
0268 23 |
0269 7E |
026A 5680 |
026C CA2B03 C 693 | JZ | NODATA |
026F EB 697 | XCHG |
0270 3600 698 | MVI | M,0 |
0272 23 699 | INX | H |
0273 3600 700 | MVI | M,0 |
0275 23 701 | INX | H |
0276 3600 702 | MVI | M,0 |
0278 2B 703 | DCX | H |
0279 2B 704 | DCX | H |
027A EB 705 | XCHG |
027B 2B 706 | DCX | H |
027C 73 707 | MOV | M,E |
027D 2B 708 | DCX | H |
<table>
<thead>
<tr>
<th>LOC</th>
<th>OBJ</th>
<th>SOURCE</th>
<th>STATEMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>27E</td>
<td>72</td>
<td>MOV D,</td>
<td>M, D</td>
</tr>
<tr>
<td>27F</td>
<td>23</td>
<td>INX B</td>
<td>H</td>
</tr>
<tr>
<td>280</td>
<td>23</td>
<td>MOV B,</td>
<td>H</td>
</tr>
<tr>
<td>281</td>
<td>46</td>
<td>INX B</td>
<td>H</td>
</tr>
<tr>
<td>282</td>
<td>23</td>
<td>MOV E,</td>
<td>H</td>
</tr>
<tr>
<td>283</td>
<td>4E</td>
<td>MOV C,</td>
<td>M</td>
</tr>
<tr>
<td>284</td>
<td>72</td>
<td>MOV B</td>
<td>; Read both length bytes to BC</td>
</tr>
<tr>
<td>285</td>
<td>23</td>
<td>INX B</td>
<td>; Save the length</td>
</tr>
<tr>
<td>286</td>
<td>09</td>
<td>LXI 0FFF7H</td>
<td>; Subtract 9 for the control bytes</td>
</tr>
<tr>
<td>287</td>
<td>72</td>
<td>DAD B</td>
<td>; leaving length of data bytes</td>
</tr>
<tr>
<td>288</td>
<td>23</td>
<td>SHLD MSGBUF</td>
<td>; Save the length</td>
</tr>
<tr>
<td>289</td>
<td>23</td>
<td>INX B</td>
<td>; point at the length bytes</td>
</tr>
<tr>
<td>290</td>
<td>13</td>
<td>INX D</td>
<td>; Set BC to the counter value</td>
</tr>
<tr>
<td>291</td>
<td>13</td>
<td>INX D</td>
<td>; The start of the message to be here</td>
</tr>
<tr>
<td>292</td>
<td>44</td>
<td>MOV B,</td>
<td>H</td>
</tr>
<tr>
<td>293</td>
<td>4D</td>
<td>MOV C,</td>
<td>L</td>
</tr>
<tr>
<td>294</td>
<td>01</td>
<td>LXI 2</td>
<td>; Save BC for later</td>
</tr>
<tr>
<td>295</td>
<td>00</td>
<td>LXI MSGBUF+2</td>
<td>; Copy (BC) bytes from (DE) to (HL)</td>
</tr>
<tr>
<td>296</td>
<td>72</td>
<td>LXI MSGBUF+2</td>
<td>; Subtract 9 for the control bytes</td>
</tr>
<tr>
<td>297</td>
<td>72</td>
<td>LXI 0FFF7H</td>
<td>; Save the length</td>
</tr>
<tr>
<td>298</td>
<td>72</td>
<td>LXI 0FFF7H</td>
<td>; Subtract 9 for the control bytes</td>
</tr>
<tr>
<td>299</td>
<td>72</td>
<td>LXI 0FFF7H</td>
<td>; Save the length</td>
</tr>
<tr>
<td>300</td>
<td>0B</td>
<td>DCX B</td>
<td>; Adjust the counter value</td>
</tr>
<tr>
<td>301</td>
<td>0B</td>
<td>DCX B</td>
<td>; Adjust the counter value</td>
</tr>
<tr>
<td>302</td>
<td>3A</td>
<td>LXI 0FFF7H</td>
<td>; Subtract 9 for the control bytes</td>
</tr>
<tr>
<td>303</td>
<td>FA</td>
<td>LXI 0FFF7H</td>
<td>; Subtract 9 for the control bytes</td>
</tr>
<tr>
<td>304</td>
<td>03</td>
<td>LXI 0FFF7H</td>
<td>; Subtract 9 for the control bytes</td>
</tr>
<tr>
<td>305</td>
<td>EF</td>
<td>MVI A,</td>
<td>TRUE</td>
</tr>
<tr>
<td>306</td>
<td>1F</td>
<td>JZ NOTPOLL</td>
<td>; The start of the message to be here</td>
</tr>
<tr>
<td>307</td>
<td>02</td>
<td>MOV B,</td>
<td>H</td>
</tr>
<tr>
<td>308</td>
<td>00</td>
<td>LXI 0FFF7H</td>
<td>; Save the length</td>
</tr>
<tr>
<td>309</td>
<td>10</td>
<td>MOV B,</td>
<td>H</td>
</tr>
<tr>
<td>310</td>
<td>00</td>
<td>LXI 0FFF7H</td>
<td>; Save the length</td>
</tr>
<tr>
<td>311</td>
<td>0B</td>
<td>DCX B</td>
<td>; Adjust the counter value</td>
</tr>
<tr>
<td>312</td>
<td>0B</td>
<td>DCX B</td>
<td>; Adjust the counter value</td>
</tr>
<tr>
<td>313</td>
<td>3A</td>
<td>LXI 0FFF7H</td>
<td>; Subtract 9 for the control bytes</td>
</tr>
<tr>
<td>314</td>
<td>FA</td>
<td>LXI 0FFF7H</td>
<td>; Subtract 9 for the control bytes</td>
</tr>
<tr>
<td>315</td>
<td>03</td>
<td>LXI 0FFF7H</td>
<td>; Subtract 9 for the control bytes</td>
</tr>
<tr>
<td>316</td>
<td>EF</td>
<td>MVI A,</td>
<td>TRUE</td>
</tr>
<tr>
<td>317</td>
<td>1F</td>
<td>JZ NOTPOLL</td>
<td>; The start of the message to be here</td>
</tr>
<tr>
<td>318</td>
<td>02</td>
<td>MOV B,</td>
<td>H</td>
</tr>
<tr>
<td>319</td>
<td>00</td>
<td>LXI 0FFF7H</td>
<td>; Save the length</td>
</tr>
<tr>
<td>320</td>
<td>10</td>
<td>MOV B,</td>
<td>H</td>
</tr>
<tr>
<td>321</td>
<td>00</td>
<td>LXI 0FFF7H</td>
<td>; Save the length</td>
</tr>
<tr>
<td>322</td>
<td>0B</td>
<td>DCX B</td>
<td>; Adjust the counter value</td>
</tr>
<tr>
<td>323</td>
<td>0B</td>
<td>DCX B</td>
<td>; Adjust the counter value</td>
</tr>
<tr>
<td>324</td>
<td>3A</td>
<td>LXI 0FFF7H</td>
<td>; Subtract 9 for the control bytes</td>
</tr>
<tr>
<td>325</td>
<td>FA</td>
<td>LXI 0FFF7H</td>
<td>; Subtract 9 for the control bytes</td>
</tr>
<tr>
<td>326</td>
<td>03</td>
<td>LXI 0FFF7H</td>
<td>; Subtract 9 for the control bytes</td>
</tr>
<tr>
<td>327</td>
<td>EF</td>
<td>MVI A,</td>
<td>TRUE</td>
</tr>
<tr>
<td>328</td>
<td>1F</td>
<td>JZ NOTPOLL</td>
<td>; The start of the message to be here</td>
</tr>
<tr>
<td>329</td>
<td>02</td>
<td>MOV B,</td>
<td>H</td>
</tr>
<tr>
<td>330</td>
<td>00</td>
<td>LXI 0FFF7H</td>
<td>; Save the length</td>
</tr>
<tr>
<td>331</td>
<td>10</td>
<td>MOV B,</td>
<td>H</td>
</tr>
<tr>
<td>332</td>
<td>00</td>
<td>LXI 0FFF7H</td>
<td>; Save the length</td>
</tr>
<tr>
<td>333</td>
<td>0B</td>
<td>DCX B</td>
<td>; Adjust the counter value</td>
</tr>
<tr>
<td>334</td>
<td>0B</td>
<td>DCX B</td>
<td>; Adjust the counter value</td>
</tr>
<tr>
<td>335</td>
<td>3A</td>
<td>LXI 0FFF7H</td>
<td>; Subtract 9 for the control bytes</td>
</tr>
<tr>
<td>336</td>
<td>FA</td>
<td>LXI 0FFF7H</td>
<td>; Subtract 9 for the control bytes</td>
</tr>
<tr>
<td>337</td>
<td>03</td>
<td>LXI 0FFF7H</td>
<td>; Subtract 9 for the control bytes</td>
</tr>
<tr>
<td>338</td>
<td>EF</td>
<td>MVI A,</td>
<td>TRUE</td>
</tr>
<tr>
<td>LOC</td>
<td>OBJ</td>
<td>LINE</td>
<td>SOURCE STATEMENT</td>
</tr>
<tr>
<td>------</td>
<td>-------</td>
<td>------</td>
<td>-----------------------------------</td>
</tr>
<tr>
<td>02AB</td>
<td>C3B002</td>
<td>739</td>
<td>JMP FILPOL</td>
</tr>
<tr>
<td>02AE</td>
<td>3E00</td>
<td>740</td>
<td>NOTPOLL:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>741</td>
<td>MVI A,FALSE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>742</td>
<td>FILPOL:</td>
</tr>
<tr>
<td>02B0</td>
<td>32490A</td>
<td>743</td>
<td>STA POLL</td>
</tr>
<tr>
<td>02B3</td>
<td>3E00</td>
<td>744</td>
<td>MVI A,FALSE</td>
</tr>
<tr>
<td>02B5</td>
<td>324A0A</td>
<td>745</td>
<td>STA BADFRAME</td>
</tr>
<tr>
<td>02B8</td>
<td>3AFA03</td>
<td>746</td>
<td>LDA MSGBUF+3</td>
</tr>
<tr>
<td>02BB</td>
<td>E6EF</td>
<td>747</td>
<td>ANI 0EFH</td>
</tr>
<tr>
<td>02BD</td>
<td>PE03</td>
<td>748</td>
<td>CPI UI</td>
</tr>
<tr>
<td>02BF</td>
<td>CAD102</td>
<td>749</td>
<td>JZ GETREST</td>
</tr>
<tr>
<td>02C2</td>
<td>PEAF</td>
<td>750</td>
<td>CPI XID</td>
</tr>
<tr>
<td>02C4</td>
<td>CAP802</td>
<td>751</td>
<td>JZ RETXID</td>
</tr>
<tr>
<td>02C7</td>
<td>FE3</td>
<td>752</td>
<td>CPI TEST</td>
</tr>
<tr>
<td>02C9</td>
<td>CA0703</td>
<td>753</td>
<td>JZ RETTEST</td>
</tr>
<tr>
<td>02CC</td>
<td>3EFF</td>
<td>754</td>
<td>MVI A,TRUE</td>
</tr>
<tr>
<td>02CE</td>
<td>324A0A</td>
<td>755</td>
<td>STA BADFRAME</td>
</tr>
<tr>
<td></td>
<td></td>
<td>756</td>
<td>GETREST:</td>
</tr>
<tr>
<td>02D1</td>
<td>78</td>
<td>757</td>
<td>MOV A,B</td>
</tr>
<tr>
<td>02D2</td>
<td>B1</td>
<td>758</td>
<td>ORA C</td>
</tr>
<tr>
<td>02D3</td>
<td>CA1402</td>
<td>759</td>
<td>JZ ENDING</td>
</tr>
<tr>
<td>02D6</td>
<td>2B</td>
<td>760</td>
<td>DCX H</td>
</tr>
<tr>
<td>02D7</td>
<td>CDA603</td>
<td>761</td>
<td>CALL COPY</td>
</tr>
<tr>
<td></td>
<td></td>
<td>762</td>
<td>ENDIN:</td>
</tr>
<tr>
<td>02DA</td>
<td>3A490A</td>
<td>763</td>
<td>LDA POLL</td>
</tr>
<tr>
<td>02DD</td>
<td>PEFF</td>
<td>764</td>
<td>CPI TRUE</td>
</tr>
<tr>
<td>02DF</td>
<td>CA2503</td>
<td>765</td>
<td>JZ EREND</td>
</tr>
<tr>
<td>02E2</td>
<td>3A4A0A</td>
<td>766</td>
<td>LDA BADFRAME</td>
</tr>
<tr>
<td>02E5</td>
<td>PEFF</td>
<td>767</td>
<td>CPI TRUE</td>
</tr>
<tr>
<td>02E7</td>
<td>CA2503</td>
<td>768</td>
<td>JZ EREND</td>
</tr>
</tbody>
</table>

- Set the poll flag
- Set the default frame flag
- Ignore the poll bit now
- UI Frame?
- XID Frame?
- Return the XID in that case
- TEST Frame?
- Return the test frame
- Unknown control byte
- Done
- Write over the control byte
- If the poll bit is set then an ERROR
- Unknown control byte
<table>
<thead>
<tr>
<th>LOC</th>
<th>OBJ</th>
<th>LINE</th>
<th>SOURCE</th>
<th>STATEMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>02EA</td>
<td>11F703</td>
<td>769</td>
<td>LXI</td>
<td>D,MSGBUF</td>
</tr>
<tr>
<td>02ED</td>
<td>3A410A</td>
<td>770</td>
<td>LDA</td>
<td>N?RXBUF</td>
</tr>
<tr>
<td>02F8</td>
<td>323F0A</td>
<td>771</td>
<td>STA</td>
<td>?RXBUF</td>
</tr>
<tr>
<td>02F3</td>
<td>3EFF</td>
<td>772</td>
<td>MVI</td>
<td>A,0FFH</td>
</tr>
<tr>
<td>02F5</td>
<td>C3303</td>
<td>773</td>
<td>JMP</td>
<td>FINIS</td>
</tr>
</tbody>
</table>

; Points at the data
; A successful read so store temp
; in permanent store
; Indicating success

774
775; Now deal with the exceptions
776
777 RETXID:

<table>
<thead>
<tr>
<th>LOC</th>
<th>OBJ</th>
<th>LINE</th>
<th>SOURCE</th>
<th>STATEMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>02F8</td>
<td>11EC03</td>
<td>778</td>
<td>LXI</td>
<td>D,XIDFRAME</td>
</tr>
<tr>
<td>02FB</td>
<td>CD3103</td>
<td>779</td>
<td>CALL</td>
<td>DESTPSET</td>
</tr>
<tr>
<td>02FE</td>
<td>AF</td>
<td>780</td>
<td></td>
<td></td>
</tr>
<tr>
<td>02FF</td>
<td>06FF</td>
<td>781</td>
<td>XRA</td>
<td>A</td>
</tr>
<tr>
<td>0301</td>
<td>CD0F01</td>
<td>782</td>
<td>MVI</td>
<td>B,0FFH</td>
</tr>
<tr>
<td>0304</td>
<td>C32503</td>
<td>783</td>
<td>CALL</td>
<td>SNDMSG</td>
</tr>
<tr>
<td>0307</td>
<td>2AF703</td>
<td>784</td>
<td>JMP</td>
<td>EREN</td>
</tr>
</tbody>
</table>

; Set up the destination address
; and set the P/F bit to the received value
; Send with no ACK
; Control frame
; Send the frame
; Finish up

785
786 RETTEST:

<table>
<thead>
<tr>
<th>LOC</th>
<th>OBJ</th>
<th>LINE</th>
<th>SOURCE</th>
<th>STATEMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>030A</td>
<td>23</td>
<td>787</td>
<td>LHL</td>
<td>MSGBUF</td>
</tr>
<tr>
<td>030B</td>
<td>22F303</td>
<td>788</td>
<td>INX</td>
<td>H</td>
</tr>
<tr>
<td>030E</td>
<td>D5</td>
<td>789</td>
<td>SHLD</td>
<td>TESTFRAME</td>
</tr>
<tr>
<td>030F</td>
<td>11F303</td>
<td>790</td>
<td>PUS</td>
<td>D</td>
</tr>
<tr>
<td>0312</td>
<td>CD3103</td>
<td>791</td>
<td>LXI</td>
<td>D,TESTFRAME</td>
</tr>
<tr>
<td>0315</td>
<td>D1</td>
<td>792</td>
<td>CALL</td>
<td>DESTPSET</td>
</tr>
<tr>
<td>0316</td>
<td>21F703</td>
<td>793</td>
<td>POP</td>
<td>D</td>
</tr>
<tr>
<td>0319</td>
<td>CD6A03</td>
<td>794</td>
<td>LXI</td>
<td>H,TESTFRAME+4</td>
</tr>
<tr>
<td>031C</td>
<td>11F303</td>
<td>795</td>
<td>CALL</td>
<td>COPY</td>
</tr>
<tr>
<td>031F</td>
<td>AF</td>
<td>796</td>
<td>LXI</td>
<td>D,TESTFRAME</td>
</tr>
</tbody>
</table>

; Read the length word
; To include the control byte
; Save present position in Rx buffer
; Set the Destination address and Poll bit
; Data to go there
; Copy the data to the Tx buffer
; Send with no ACK
; Control frame
<table>
<thead>
<tr>
<th>LOC</th>
<th>OBJ</th>
<th>LINE</th>
<th>SOURCE</th>
<th>STATEMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0322</td>
<td>CD0F01</td>
<td>799</td>
<td>CALL SNDMSG</td>
<td>; Send the frame</td>
</tr>
<tr>
<td></td>
<td></td>
<td>800</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>801</td>
<td>; Error exit, also used by control frames</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>802</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0325</td>
<td>3A410A</td>
<td>804</td>
<td>LDA N?RXBUF</td>
<td>; Adjust the pointers</td>
</tr>
<tr>
<td>0328</td>
<td>323F0A</td>
<td>805</td>
<td>STA ?RXBUF</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>806</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>807</td>
<td>; Exit point if no data</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>808</td>
<td></td>
<td></td>
</tr>
<tr>
<td>032B</td>
<td>CD0101</td>
<td>810</td>
<td>CALL DLAY</td>
<td>; Wait for TAC to update Regs</td>
</tr>
<tr>
<td>032E</td>
<td>3E00</td>
<td>811</td>
<td>MVI A,00</td>
<td>; Unsuccessful</td>
</tr>
<tr>
<td></td>
<td></td>
<td>812</td>
<td>FINIS:</td>
<td></td>
</tr>
<tr>
<td>0330</td>
<td>C9</td>
<td>813</td>
<td>RET</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>814</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>815</td>
<td>$EJECT</td>
<td></td>
</tr>
</tbody>
</table>
Set the Poll bit and Destination address

FUNCTION: DESTPSET

INPUTS:
- DE = Frame to be adjusted
- POLLB = True if F bit to be set
- = False if F bit to be clear
- MSGBUF+2 = Destination address

OUTPUTS: None

DESTROYS: A, Flags

DESCRIPTION:
Sets the poll bit to the value it was in the received frame, and then sets the destination address from MSGBUF.

DESTPSET:

0331 D5 833 PUSH D ; Save status
0332 C5 834 PUSH B ;
0333 3AF903 C 835 LDA MSGBUF+2 ; Get the destination address
0336 13 836 INX D ;
0337 13 837 INX D ; Point at Destination address
0338 12 838 STAX D ;
0339 13 839 INX D ; Point at control byte
033A 3A490A C 840 LDA POLL ; Poll bit set?
033D PEFF 841 CPI TRUE ;
033F 1A 842 LDAX D ; Read control byte
0340 C24903 C 843 JNZ NOSET ;
0343 0610 844 MVI B,POLLB ; Set up the poll bit
0345 B0 845 ORA B ;
<table>
<thead>
<tr>
<th>LOC</th>
<th>OBJ</th>
<th>LINE</th>
<th>SOURCE</th>
<th>STATEMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0346</td>
<td>C34C03</td>
<td>846</td>
<td>JMP</td>
<td>POLRET; Save it</td>
</tr>
<tr>
<td></td>
<td></td>
<td>847</td>
<td>NOSET:</td>
<td></td>
</tr>
<tr>
<td>0349</td>
<td>06EF</td>
<td>848</td>
<td>MVI</td>
<td>B,NOPOLL;</td>
</tr>
<tr>
<td>034B</td>
<td>A0</td>
<td>849</td>
<td>ANA</td>
<td>B;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>850</td>
<td>POLRET:</td>
<td></td>
</tr>
<tr>
<td>034C</td>
<td>12</td>
<td>851</td>
<td>STAX</td>
<td>D; Save the value</td>
</tr>
<tr>
<td>034D</td>
<td>C1</td>
<td>852</td>
<td>POP</td>
<td>B;</td>
</tr>
<tr>
<td>034E</td>
<td>D1</td>
<td>853</td>
<td>POP</td>
<td>D;</td>
</tr>
<tr>
<td>034F</td>
<td>C9</td>
<td>854</td>
<td>RET</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>855</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>856</td>
<td>$EJECT</td>
<td></td>
</tr>
</tbody>
</table>
FUNCTION: STAT
INPUTS: None
OUTPUTS: DE = buffer containing 16 registers and 11 Event counters
CALLS: None
DESCRIPTION: This routine returns a copy of the 16 registers and the 11 Event counters in a buffer pointed to by DE.

LOC  OBJ  LINE  SOURCE STATEMENT
0350  11F703  C  870  LXI  D,MSGBUF  ; Get the buffer address
0353  DB70   871  IN  70H  ; Read the Status registers into the buffer
0355  12    872  STAX  D  
0356  13    873  INX  D  
0357  DB71   874  IN  71H  
0359  12    875  STAX  D  
035A  13    876  INX  D  
035B  DB72   877  IN  72H  
035D  12    878  STAX  D  
035E  13    879  INX  D  
035F  DB73   880  IN  73H  
0361  12    881  STAX  D  
0362  13    882  INX  D  
0363  DB74   883  IN  74H  
0365  12    884  STAX  D  
0366  13    885  INX  D  
0367  DB75   886  IN  75H  

<table>
<thead>
<tr>
<th>LOC</th>
<th>OBJ</th>
<th>STATEMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0369</td>
<td>12</td>
<td>STAX D</td>
</tr>
<tr>
<td>036A</td>
<td>13</td>
<td>INX 76H</td>
</tr>
<tr>
<td>036B</td>
<td>12</td>
<td>STAX D</td>
</tr>
<tr>
<td>036C</td>
<td>13</td>
<td>INX 77H</td>
</tr>
<tr>
<td>036D</td>
<td>12</td>
<td>STAX D</td>
</tr>
<tr>
<td>036E</td>
<td>13</td>
<td>INX 78H</td>
</tr>
<tr>
<td>036F</td>
<td>12</td>
<td>STAX D</td>
</tr>
<tr>
<td>0370</td>
<td>13</td>
<td>INX 79H</td>
</tr>
<tr>
<td>0371</td>
<td>12</td>
<td>STAX D</td>
</tr>
<tr>
<td>0372</td>
<td>13</td>
<td>INX 7AH</td>
</tr>
<tr>
<td>0373</td>
<td>12</td>
<td>STAX D</td>
</tr>
<tr>
<td>0374</td>
<td>13</td>
<td>INX 7BH</td>
</tr>
<tr>
<td>0375</td>
<td>12</td>
<td>STAX D</td>
</tr>
<tr>
<td>0376</td>
<td>13</td>
<td>INX 7CH</td>
</tr>
<tr>
<td>0377</td>
<td>12</td>
<td>STAX D</td>
</tr>
<tr>
<td>0378</td>
<td>13</td>
<td>INX 7DH</td>
</tr>
<tr>
<td>0379</td>
<td>12</td>
<td>STAX D</td>
</tr>
<tr>
<td>037A</td>
<td>13</td>
<td>INX 7FH</td>
</tr>
<tr>
<td>LOC</td>
<td>OBJ</td>
<td>LINE</td>
</tr>
<tr>
<td>-------</td>
<td>-----</td>
<td>------</td>
</tr>
<tr>
<td>0391</td>
<td>12</td>
<td>917</td>
</tr>
<tr>
<td>0392</td>
<td>13</td>
<td>918</td>
</tr>
<tr>
<td>0393</td>
<td>213405</td>
<td>920</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0396</td>
<td>060B</td>
<td>921</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0398</td>
<td>7E</td>
<td>923</td>
</tr>
<tr>
<td>0399</td>
<td>3600</td>
<td>924</td>
</tr>
<tr>
<td>039B</td>
<td>12</td>
<td>925</td>
</tr>
<tr>
<td>039C</td>
<td>23</td>
<td>926</td>
</tr>
<tr>
<td>039D</td>
<td>13</td>
<td>927</td>
</tr>
<tr>
<td>039E</td>
<td>05</td>
<td>928</td>
</tr>
<tr>
<td>039F</td>
<td>C29803</td>
<td>929</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>03A2</td>
<td>11F703</td>
<td>931</td>
</tr>
<tr>
<td>03A5</td>
<td>C9</td>
<td>932</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>933</td>
<td></td>
<td>934</td>
</tr>
</tbody>
</table>

CJ: ]
FUNCTION: COPY

INPUTS: BC = counter
        DE = Source address
        HL = Destination address

OUTPUTS: None

DESTROYS: A, B, C, D, E, H, L, Flags

DESCRIPTION:
Copies the (BC) bytes of data starting from (DE) and
writing them to memory starting from (HL).

COPY:

LDA X D ; Read the data
MOV M,A ; Store it in destination
INX D ; Adjust the pointers
INX H ;
DCX B ; End of the data copy?
MOV A,B ;
ORA C ;
JNZ COPY ; NZ = not yet complete so loop
RET ;
LOC   OBJ  LINE  SOURCE STATEMENT

961 ;-----------------------------------------------
962 ; Messages
963 ;-----------------------------------------------

03B1 4475706C 964 FAILMSG: DB 'Duplicate TAC addresses',CR,LF,0
69636174
65205441
43206164
64726573
7365730D
0A00

03CB 54414320 965 READY: DB 'TAC initialised',CR,LF,0
696E6974
69616C69
7365640D
0A00

03DD 54414320 966 OUTITIS: DB 'TAC isolated',CR,LF,0
69736F6C
61746564
0D0A00

967

968 ;-----------------------------------------------
969 ; Local Data Segment
970 ;-----------------------------------------------

03EC 0400 971 XIDFRAME: ; XID response
03EE 00 972 DW 4 ; Length field
03EF AF 973 DB 0 ; Destination (to be filled in)
03F0 810102 974 DB XID ; XID Control byte (adjust F bit = cmd P bit)
975 DB 81H,01H,02H ; XID I Field = Class I LLC, window size - 1
976 ;
977 TESTFRAME: ; TEST response
<table>
<thead>
<tr>
<th>LOC</th>
<th>OBJ</th>
<th>LINE</th>
<th>SOURCE STATEMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>03F3</td>
<td>0000</td>
<td>978</td>
<td>DW 0</td>
</tr>
<tr>
<td>03F5</td>
<td>00</td>
<td>979</td>
<td>DB 0</td>
</tr>
<tr>
<td>03F6</td>
<td>E3</td>
<td>980</td>
<td>DB TEST</td>
</tr>
<tr>
<td></td>
<td></td>
<td>981</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>982</td>
<td>; Length field (to be filled in)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>983</td>
<td>; Destination address (to be filled in)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>984</td>
<td>; TEST control byte</td>
</tr>
<tr>
<td>03F7</td>
<td></td>
<td>985</td>
<td>; Use MSGBUF for I field</td>
</tr>
<tr>
<td></td>
<td></td>
<td>986</td>
<td>; Area for returned data</td>
</tr>
<tr>
<td></td>
<td></td>
<td>987</td>
<td></td>
</tr>
<tr>
<td>052F</td>
<td></td>
<td>988</td>
<td>CBLOCK:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>989</td>
<td>DS 2</td>
</tr>
<tr>
<td>0531</td>
<td></td>
<td>990</td>
<td>DS 2</td>
</tr>
<tr>
<td>0533</td>
<td></td>
<td>991</td>
<td>DS 1</td>
</tr>
<tr>
<td>0534</td>
<td></td>
<td>992</td>
<td>DS 11</td>
</tr>
<tr>
<td></td>
<td></td>
<td>993</td>
<td>; Control block</td>
</tr>
<tr>
<td></td>
<td></td>
<td>994</td>
<td>RXBUFl:</td>
</tr>
<tr>
<td>053F</td>
<td></td>
<td>995</td>
<td>DS 2</td>
</tr>
<tr>
<td>0541</td>
<td></td>
<td>996</td>
<td>DS 318</td>
</tr>
<tr>
<td></td>
<td></td>
<td>997</td>
<td>RXBUF2:</td>
</tr>
<tr>
<td>067F</td>
<td></td>
<td>998</td>
<td>DS 2</td>
</tr>
<tr>
<td>0681</td>
<td></td>
<td>999</td>
<td>DS 318</td>
</tr>
<tr>
<td>1000</td>
<td></td>
<td>1000</td>
<td>TXBUFl:</td>
</tr>
<tr>
<td>07BF</td>
<td></td>
<td>1001</td>
<td>DS 2</td>
</tr>
<tr>
<td>07C1</td>
<td></td>
<td>1002</td>
<td>DS 318</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1003</td>
<td>TXBUF2:</td>
</tr>
<tr>
<td>08FF</td>
<td></td>
<td>1004</td>
<td>DS 2</td>
</tr>
<tr>
<td>0901</td>
<td></td>
<td>1005</td>
<td>DS 318</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1006</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1007</td>
<td>; General Flag storage</td>
</tr>
<tr>
<td>LOC</td>
<td>OBJ</td>
<td>LINE</td>
<td>SOURCE STATEMENT</td>
</tr>
<tr>
<td>-----</td>
<td>-----</td>
<td>------</td>
<td>-----------------</td>
</tr>
<tr>
<td>1008</td>
<td>A3F</td>
<td>1009</td>
<td>?RXBUF: ; Buffer pointers</td>
</tr>
<tr>
<td>1010</td>
<td>A40</td>
<td>1011</td>
<td>?TXBUF: ;</td>
</tr>
<tr>
<td>1012</td>
<td>A41</td>
<td>1013</td>
<td>N?RXBUF: ; Temporary storage</td>
</tr>
<tr>
<td>1015</td>
<td>A42</td>
<td>1016</td>
<td>MYID: ; This node's address</td>
</tr>
<tr>
<td>1018</td>
<td>A43</td>
<td>1019</td>
<td>PRESSTAT: ; Storage for CR0 status</td>
</tr>
<tr>
<td>1020</td>
<td>A44</td>
<td>1021</td>
<td>LONG: ; Tx Frame &gt; 312 Bytes =&gt; TRUE</td>
</tr>
<tr>
<td>1022</td>
<td>A45</td>
<td>1023</td>
<td>LEN: ; Bytes left in Tx frame</td>
</tr>
<tr>
<td>1024</td>
<td>A46</td>
<td>1025</td>
<td>SNDWITHACK: ; Acknowledge flag</td>
</tr>
<tr>
<td>1026</td>
<td>A47</td>
<td>1027</td>
<td>CTLFR: ; Control frame flag</td>
</tr>
<tr>
<td>1028</td>
<td>A48</td>
<td>1029</td>
<td>POLL: ; Poll bit flag</td>
</tr>
<tr>
<td>1030</td>
<td>A49</td>
<td>1031</td>
<td>BADFRAME: ; Bad frame control word</td>
</tr>
<tr>
<td>1032</td>
<td>A4A</td>
<td>1033</td>
<td>DEST: ; Tx frame Destination ID</td>
</tr>
<tr>
<td>1034</td>
<td>A4B</td>
<td>1035</td>
<td>LASTBYTE: ; Store last byte read</td>
</tr>
<tr>
<td>1036</td>
<td>A4C</td>
<td>1037</td>
<td>BEGBUF: ; Store for Beginning of block pointer</td>
</tr>
<tr>
<td>1038</td>
<td>A4E</td>
<td>1039</td>
<td></td>
</tr>
<tr>
<td>LOC</td>
<td>OBJ</td>
<td>LINE</td>
<td>SOURCE STATEMENT</td>
</tr>
<tr>
<td>-----</td>
<td>-----</td>
<td>------</td>
<td>---------------------------------------</td>
</tr>
<tr>
<td>0A50</td>
<td></td>
<td>1037</td>
<td>ADDUP: ; Address duplication counter</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1038</td>
<td>DS 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1039</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>1040</td>
<td>; Stack Area</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1041</td>
<td></td>
</tr>
<tr>
<td>0A51</td>
<td></td>
<td>1042</td>
<td>DS 30</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1043</td>
<td>NEWSP: ;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1044</td>
<td>END</td>
</tr>
</tbody>
</table>

0 error(s) detected
APPENDIX F

SNIOS for the TAC Network
PROGRAM = TOKSNIOS.ASM
LANGUAGE = 8085 Assembler Code
TARGET SYSTEM = UCT SABus Kits
AUTHOR = Q.P. Mc Grath
DATE = 13 August 1985

TOKSNIOS is the user written program which forms the link
between the local CP/M system and the network. The data is
transmitted, under this programs control, across the network
and to and from the Network I/O System (NIOS), which is the
link to the local CP/M system.

Once initialisation is complete, TOKSNIOS awaits data from the
NIOS. The NIOS fills the buffer with data in the format below:

The data format to and from the NIOS is:
FMT - The format code. For CP/NET 1.2 there is only one possible format pair, 00H for transmission of data and 01H for returned data. This allows 1 byte for each of the Destination ID code, the Source ID code Function, Size field, and 256 bytes for the data field.

DID - The Destination ID code field.

SID - Source ID code field.

FNC - Function code field.

SIZ - Size field.

DB0 - First data byte field

DB1 - Second data byte field

... 

DBn - Nth data byte field. (to a maximum of 256 bytes)

The above data is then adjusted slightly to fit into the format required by the TAC User Routines (TACUR) as shown below, and transmitted across the network using the TACUR program:

LENGTH (H) - The LSB length byte. (calculated from the SIZ byte)

LENGTH (L) - The MSB length byte.

DA - The destination ID code which exactly the same as the DID above.

Data - The data field. In this section the FNC and SIZ bytes are added to the data bytes.

The reverse occurs with the incoming data, where it is converted from the TAC format to the NIOS format. (The FMT code is assumed to be 00H or 01H and is not transferred across the network.)
The Requester is always the initiator in the transmission of data and so the sequence SEND data RECEIVE data is always followed.
LOC  OBJ  LINE  SOURCE STATEMENT

68 ;================================================================
69 ;
70 ;  EQUATES
71 ;
72 ;================================================================
73 ;
74 ;--------- True or False Equates
75 ;
76 ;---------
77 

0000
FF00

78 FALSE EQU 0
79 TRUE EQU NOT FALSE
80 

0001
007F

81 SERVERID EQU 01H
82 MA EQU 7FH
83 

84 ;--------- Network Status Byte Equates
85 ;
86 ;---------
87 

0010
0002
0001

88 ACTIVE EQU 00010000B ; slave logged in
89 RCVERR EQU 00000100B ; error in received msg
90 SNDERR EQU 0000001B ; unable to send message
91 

92 ;--------- General Equates
93 

94 ;
95 

000A
000D

96 LF EQU 0AH ; Line Feed
97 CR EQU 0DH ; Carriage Return
LOC OBJ LINE SOURCE STATEMENT

0064 0064 98 RETRYCNT EQU 100 ; Retry counter for receive message
00CC 00CC 99 DLOCOUNT EQU 204 ; Counter for the delay loop

101 102 ;--------------------------------------------------- BDOS Call equates
103 ;---------------------------------------------------
104 ;---------------------------------------------------
105 106 BDOS EQU 0005H ; BDOS call address
107 108 LOGIN EQU 64 ; Login NDOS function
109 PRINTSTRING EQU 9 ; Bdos call for string print

110 111 ;--------------------------------------------------- TACUR Equates
112 ;---------------------------------------------------
113 ;---------------------------------------------------
114
115 NETINIT EQU 0D008H ; TAC and network init routine
116 SNDMSG EQU 0D003H ; Transmission routine
117 RECMMSG EQU 0D006H ; Receive routine
118
119 $EJECT
Jump vector for SNIOS entry points

Jump vector for network initialization
Jump vector for network status
Jump vector for return config table addr
Jump vector for send message on network
Jump vector for receive message from network
Jump vector for network error
Jump vector for network warm boot

Local Data Segment

Slave Configuration Table

network status byte
slave processor ID number
A: Disk device
B: C: D: E:

00000000B
1
0,0
0,0
0,0
0,0

0015 00 0017 0000 0019 0000 001D 0000 001F 0000
<table>
<thead>
<tr>
<th>LOC</th>
<th>OBJ</th>
<th>LINE</th>
<th>SOURCE STATEMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0021</td>
<td>0000</td>
<td>150</td>
<td>DB  0,0          ; F: &quot;</td>
</tr>
<tr>
<td>0023</td>
<td>0000</td>
<td>151</td>
<td>DB  0,0          ; G: &quot;</td>
</tr>
<tr>
<td>0025</td>
<td>0000</td>
<td>152</td>
<td>DB  0,0          ; H: &quot;</td>
</tr>
<tr>
<td>0027</td>
<td>0000</td>
<td>153</td>
<td>DB  0,0          ; I: &quot;</td>
</tr>
<tr>
<td>0029</td>
<td>0000</td>
<td>154</td>
<td>DB  0,0          ; J: &quot;</td>
</tr>
<tr>
<td>002B</td>
<td>0000</td>
<td>155</td>
<td>DB  0,0          ; K: &quot;</td>
</tr>
<tr>
<td>002D</td>
<td>0000</td>
<td>156</td>
<td>DB  0,0          ; L: &quot;</td>
</tr>
<tr>
<td>002F</td>
<td>0000</td>
<td>157</td>
<td>DB  0,0          ; M: &quot;</td>
</tr>
<tr>
<td>0031</td>
<td>0000</td>
<td>158</td>
<td>DB  0,0          ; N: &quot;</td>
</tr>
<tr>
<td>0033</td>
<td>0000</td>
<td>159</td>
<td>DB  0,0          ; O: &quot;</td>
</tr>
<tr>
<td>0035</td>
<td>0000</td>
<td>160</td>
<td>DB  0,0          ; P: &quot;</td>
</tr>
<tr>
<td>160</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0037</td>
<td>0000</td>
<td>161</td>
<td>DB  0,0          ; console device</td>
</tr>
<tr>
<td>163</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0039</td>
<td>0000</td>
<td>164</td>
<td>DB  0,0          ; list device:</td>
</tr>
<tr>
<td>003B</td>
<td>0000</td>
<td>165</td>
<td>DB  0            ; buffer index</td>
</tr>
<tr>
<td>003C</td>
<td>0000</td>
<td>166</td>
<td>DB  0            ; FMT</td>
</tr>
<tr>
<td>003D</td>
<td>0000</td>
<td>167</td>
<td>DB  SERVERID    ; DID</td>
</tr>
<tr>
<td>003E</td>
<td>0000</td>
<td>168</td>
<td>SLVID1: DS  1    ; SID</td>
</tr>
<tr>
<td>003F</td>
<td>0000</td>
<td>169</td>
<td>DB  5            ; FNC</td>
</tr>
<tr>
<td>0040</td>
<td>0000</td>
<td>170</td>
<td>DB  0            ; SIZ</td>
</tr>
<tr>
<td>0041</td>
<td>0000</td>
<td>171</td>
<td>DS  1            ; MSG(0) List number</td>
</tr>
<tr>
<td>0042</td>
<td>0000</td>
<td>172</td>
<td>DS  255          ; MSG(1) ... MSG(255)</td>
</tr>
<tr>
<td>173</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>174</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>175</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>176</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>177</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0141</td>
<td>0141</td>
<td>178</td>
<td>DS  320          ; Buffer</td>
</tr>
<tr>
<td>179</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LOC</td>
<td>OBJ</td>
<td>LINE</td>
<td>SOURCE STATEMENT</td>
</tr>
<tr>
<td>--------</td>
<td>-------</td>
<td>--------</td>
<td>------------------</td>
</tr>
<tr>
<td>0281</td>
<td>0000</td>
<td>180</td>
<td>;-----------------------------------------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>181</td>
<td>Storage for Message address from NDOS</td>
</tr>
<tr>
<td></td>
<td></td>
<td>182</td>
<td>;-----------------------------------------------</td>
</tr>
<tr>
<td>0283</td>
<td></td>
<td>183</td>
<td>MSGADR:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>184</td>
<td>DW 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>185</td>
<td>SLAVEID:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>186</td>
<td>DS 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>187</td>
<td>; Save for My Address</td>
</tr>
<tr>
<td></td>
<td></td>
<td>188</td>
<td>;-----------------------------------------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>189</td>
<td>Warm Boot Message</td>
</tr>
<tr>
<td></td>
<td></td>
<td>190</td>
<td>;-----------------------------------------------</td>
</tr>
<tr>
<td>0284</td>
<td>0D3C4350</td>
<td>191</td>
<td>WBOOTMSG:</td>
</tr>
<tr>
<td>2F4E4554</td>
<td></td>
<td>192</td>
<td>DB CR,'&lt;CP/NET&gt;'</td>
</tr>
<tr>
<td>3E</td>
<td></td>
<td>193</td>
<td>DB '$'</td>
</tr>
<tr>
<td>028D</td>
<td>24</td>
<td>194</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>195</td>
<td>$EJECT</td>
</tr>
</tbody>
</table>
LOC OBJ LINE SOURCE STATEMENT

196 ;============================================================
197 ;
198 ;
199 ;
200 ;============================================================
201 ;
202 ; NAME : NTWRKINIT
203 ; INPUTS : None
204 ; OUTPUTS : A - containing 0 if initialisation was successful
205 ; - containing 0FFH if unsuccessful
206 ; CALLS : External NETINIT
207 ; DESTROYS : None
208 ; DESCRIPTION : NTWRKINIT sets up the TAC's parameters and then
209 ; checks the network for activity. If activity is
210 ; found then it will check for duplicate addresses
211 ; if found it will tell the user and return with A
212 ; set to FFH, if no duplication of addresses occurs
213 ; then it will enable transmission. Should the
214 ; network be found to be dead, then it will wait for
215 ; activity and then check for duplicate addresses.
216 ; In the event of no duplicated addresses the TAC is
217 ; enabled for transmission and the A register is set
218 ; to 0H.
219 ;
220 ;-----------------------------------
221 NTWRKINIT:
222
223 PUSH B ; Save registers
224 PUSH D ;
225 PUSH H ;
TAC SLAVE NETWORK I/O SYSTEM

LOC  OBJ  LINE  SOURCE STATEMENT

0291  CD00D0  226  CALL  NETINIT ; Call the network init program in TACUR
0294  D29C02  227  JNC  TACOK ; If no carry then init was successful
0297  3EFF   228  MVI  A,0FFH ; Else there was an error
0299  C3AF02  229  JMP  ENDIT ;

230
231 ; This means a successful initialisation
232
233 TACOK:
234  IN    MA ; Read my address
029C  DB7F  235  STA   SLVID ; Store in frame
029E  321600  236  STA   SLVID1 ;
02A1  323E00  237  STA   SLAVEID ; For later
02A4  328302  238  MVI  A,ACTIVE ; Set the network status byte
02A7  3E10   239  LXI   H,NETWORKSTATUS ;
02A9  211500  240  ORA   M ;
02AC  B6     241  MOV   M,A ;
02AD  77     242  XRA   A ; Return code is 0=success
02AE  AF     243  ENDIT: ;
02AF  E1     244  POP   H ; Restore registers
02B0  D1     245  POP   D ;
02B1  C1     246  POP   B ;
02B2  C9     247  RET ;
02B4  248
02B5  249
02B6  250 $EJECT
SEND MESSAGE ON THE NETWORK

SENDMSG: SENDMSG takes the message that is written in the address given by BC and writes it into the form required by TACUR. Control is then given to TACUR which sends the frame.

269 MOV H, B ; HL = message address
270 MOV L,C ; Save the message address
271 SHLD MSGADR
272 PUSH B 
273 SML10:
274 CALL RECMSG ; Flush the receive buffer
275 ORA A ; 0 = flush complete as no data read
276 JNZ SML10 ; Or loop
277
278 POP B 
279 INX B ; Point at the size field
280 INX B ;
LOC OBJ LINE SOURCE STATEMENT

02C3 03 281 INX B ;
02C4 03 282 INX B ;
02C5 0A 283 LDAX B ;
02C6 6F 284 MOV L,A ; Add on the extra control fields
02C7 2600 285 MVI H,0 ;
02C9 23 286 INX H ;
02CA 23 287 INX H ;
02CB 23 288 INX H ; (HL) = Length plus the control bytes
02CC 224101 289 SHLD DBUFF ;
02CF 0B 290 DCX B ;
02D0 0B 291 DCX B ;
02D1 0B 292 DCX B ; BC point at DID in the NDOS frame
02D2 114301 293 LXI D, DBUFF+2 ; Start of the data buffer
294 ;
295 ; Copy the data into the transmission buffer
296 ;
02D5 0A 297 LDAX B ; Read destination address
02D6 12 298 STAX D ;
02D7 03 299 INX B ; Read past source ID
02D8 03 300 INX B ;
02D9 13 301 INX D ;
02DA 2B 302 DCX H ;
02DB 7C 303 MOV A,H ; End yet?
02DC B5 304 ORA L ;
02DD CAEA02 305 JZ DLEND ;
306 DLOOP:
02E0 0A 307 LDAX B ; Read character
02E1 12 308 STAX D ; Store it
02E2 03 309 INX B ;
02E3 13 310 INX D ;
<table>
<thead>
<tr>
<th>LOC</th>
<th>OBJ</th>
<th>LINE</th>
<th>SOURCE</th>
<th>STATEMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>02E4</td>
<td>2B</td>
<td>311</td>
<td>DCX H</td>
<td>; Length at an end?</td>
</tr>
<tr>
<td>02E5</td>
<td>7C</td>
<td>312</td>
<td>MOV A,H</td>
<td>;</td>
</tr>
<tr>
<td>02E6</td>
<td>B5</td>
<td>313</td>
<td>ORA L</td>
<td>;</td>
</tr>
<tr>
<td>02E7</td>
<td>C2E002</td>
<td>314</td>
<td>JNZ DLOOP</td>
<td>;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>315</td>
<td></td>
<td>;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>316</td>
<td>; Now send it using the TACUR SNDMSG</td>
<td></td>
</tr>
<tr>
<td>02EA</td>
<td>114101</td>
<td>319</td>
<td>LXI D, DBUFF</td>
<td>; Set up pointer</td>
</tr>
<tr>
<td>02ED</td>
<td>3EFF</td>
<td>320</td>
<td>MVI A, $0FFH</td>
<td>; Set up flags</td>
</tr>
<tr>
<td>02EF</td>
<td>0600</td>
<td>321</td>
<td>MVI B, $0</td>
<td>; for send with ACKs and not control</td>
</tr>
<tr>
<td>02F1</td>
<td>CD03D0</td>
<td>322</td>
<td>CALL SNDMSG</td>
<td></td>
</tr>
<tr>
<td>02F4</td>
<td>PE00</td>
<td>323</td>
<td>CPI $0</td>
<td>; Successful?</td>
</tr>
<tr>
<td>02F6</td>
<td>CA0503</td>
<td>324</td>
<td>JZ TXSUC</td>
<td></td>
</tr>
<tr>
<td>02FA</td>
<td>3E01</td>
<td>325</td>
<td>MVI A, SNDERR</td>
<td>; Set the status byte</td>
</tr>
<tr>
<td>02FD</td>
<td>21500</td>
<td>326</td>
<td>LXI H, NETWORKSTATUS</td>
<td></td>
</tr>
<tr>
<td>02FE</td>
<td>B6</td>
<td>327</td>
<td>ORA M</td>
<td>;</td>
</tr>
<tr>
<td>02FF</td>
<td>77</td>
<td>328</td>
<td>MOV M, A</td>
<td>;</td>
</tr>
<tr>
<td>0300</td>
<td>3EFF</td>
<td>329</td>
<td>MVI A, $0FFH</td>
<td>; No - return $0FF as indication</td>
</tr>
<tr>
<td>0302</td>
<td>C31003</td>
<td>330</td>
<td>JMP TXS1</td>
<td>;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>331</td>
<td>TXSUC:</td>
<td></td>
</tr>
<tr>
<td>0305</td>
<td>3E01</td>
<td>332</td>
<td>MVI A, SNDERR</td>
<td>; Set the status byte</td>
</tr>
<tr>
<td>0307</td>
<td>EEFF</td>
<td>333</td>
<td>XRI $0FF</td>
<td></td>
</tr>
<tr>
<td>0309</td>
<td>211500</td>
<td>334</td>
<td>LXI H, NETWORKSTATUS</td>
<td></td>
</tr>
<tr>
<td>030C</td>
<td>A6</td>
<td>335</td>
<td>ANA M</td>
<td></td>
</tr>
<tr>
<td>030D</td>
<td>77</td>
<td>336</td>
<td>MOV M,A</td>
<td>;</td>
</tr>
<tr>
<td>030E</td>
<td>3E00</td>
<td>337</td>
<td>MVI A, $0</td>
<td>;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>338</td>
<td>TXS1:</td>
<td></td>
</tr>
<tr>
<td>0310</td>
<td>2A8102</td>
<td>339</td>
<td>LHLD MSGADR</td>
<td>; Restore the message address</td>
</tr>
<tr>
<td>0313</td>
<td>44</td>
<td>340</td>
<td>MOV B, H</td>
<td>;</td>
</tr>
</tbody>
</table>
### RECEIVE A MESSAGE FROM THE NETWORK

{ref to image: LOC OBJ LINE SOURCE STATEMENT}

<table>
<thead>
<tr>
<th>LOC</th>
<th>OBJ</th>
<th>LINE</th>
<th>STATEMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0316</td>
<td>60</td>
<td>346</td>
<td>;---------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| 0317 | 69  | 347  | ;
| 0318 | 228102 | 348  | ;
|      |     | 349  | ;
|      |     | 350  | ;---------------------------------------------------------------|
|      |     | 351  | ;
|      |     | 352  | ; NAME : RECEIVEMSG
|      |     | 353  | ; INPUTS : BC containing the address in which to place message
|      |     | 354  | ; OUTPUTS : Message to the buffer given
|      |     | 355  | ; CALLS : External TACUR RECMMSG
|      |     | 356  | ; DESTROYS : A, B, C, H, L, Flags
|      |     | 357  | ; DESCRIPTION : ReceiveMsg polls the TACUR RECMMSG for an input and
|      |     | 358  | ; then places the frame in the address specified by
|      |     | 359  | ; the address in BC.
|      |     | 360  | 
|      |     | 361  | ;---------------------------------------------------------------|
|      |     | 362  | ; RECEIVEMSG:
| 031B | 216400 | 363  | MOV H,B ;
|      |      | 364  | MOV L,C ;
|      |      | 365  | SHLD MSGADR ; Store for return
|      |      | 366  | 
|      |      | 367  | ; Wait for a frame
| 031E | CD06D0 | 368  | RXLOOP:
|      |      | 369  | LXI H, RETRYCNT ;
|      |      | 370  | RX LOOP: ;
| 0321 | FE00  | 371  | CALL RECMMSG ; RECMMSG returns 0 if no frame
| 0323 | C22F03 | 372  | CPI 0 ;
| 0326 | 2B  | 373  | JNZ RXL01 ; <>0 means frame received
| 0327 | 7C  | 374  | DCX H ; Check on the retry count
|      |      | 375  | MOV A,H ;


; A frame was received
RXL01:
LDAX D
MOV L,A
INX D
LDAX D
INX D
MOV H,A
(ML) = length of frame

; Set up the FORMAT byte
MVI A,01H
STAX B
INX B
LDA SLAVEID
STAX B
INX D
STAX B
INX B

; Write data
LDA D
STAX B
INX B
DCX H

; End of Data?
<table>
<thead>
<tr>
<th>LOC</th>
<th>OBJ</th>
<th>LINE</th>
<th>SOURCE</th>
<th>STATEMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0345</td>
<td>7C</td>
<td>406</td>
<td>MOV</td>
<td>A, H</td>
</tr>
<tr>
<td>0346</td>
<td>B5</td>
<td>407</td>
<td>ORA</td>
<td>L</td>
</tr>
<tr>
<td>0347</td>
<td>C24003</td>
<td>408</td>
<td>JNZ</td>
<td>RXL10</td>
</tr>
<tr>
<td>034A</td>
<td>3E02</td>
<td>409</td>
<td>MVI</td>
<td>A, RCVERR</td>
</tr>
<tr>
<td>034C</td>
<td>EEFF</td>
<td>410</td>
<td>MVI</td>
<td>A, 0FFH</td>
</tr>
<tr>
<td>034E</td>
<td>211500</td>
<td>411</td>
<td>XRI</td>
<td>0FFH</td>
</tr>
<tr>
<td>0351</td>
<td>A6</td>
<td>412</td>
<td>LXI</td>
<td>H, NETWORKSTATUS</td>
</tr>
<tr>
<td>0352</td>
<td>77</td>
<td>413</td>
<td>ANA</td>
<td>M</td>
</tr>
<tr>
<td>0353</td>
<td>3E00</td>
<td>414</td>
<td>MOV</td>
<td>M, A</td>
</tr>
<tr>
<td>0355</td>
<td>2A8102</td>
<td>415</td>
<td>MVI</td>
<td>A, 00H</td>
</tr>
<tr>
<td>0358</td>
<td>C36703</td>
<td>416</td>
<td>LXI</td>
<td>H, NETWORKSTATUS</td>
</tr>
<tr>
<td>0359</td>
<td>3E02</td>
<td>417</td>
<td>MVI</td>
<td>A, RCVERR</td>
</tr>
<tr>
<td>035D</td>
<td>211500</td>
<td>418</td>
<td>LXI</td>
<td>H, NETWORKSTATUS</td>
</tr>
<tr>
<td>0360</td>
<td>B6</td>
<td>419</td>
<td>ORA</td>
<td>M</td>
</tr>
<tr>
<td>0361</td>
<td>77</td>
<td>420</td>
<td>MOV</td>
<td>M, A</td>
</tr>
<tr>
<td>0362</td>
<td>CD6803</td>
<td>421</td>
<td>CALL</td>
<td>NTWRKERROR</td>
</tr>
<tr>
<td>0365</td>
<td>3EFF</td>
<td>422</td>
<td>MVI</td>
<td>A, 0FFH</td>
</tr>
<tr>
<td>0367</td>
<td>C9</td>
<td>423</td>
<td>RET</td>
<td></td>
</tr>
<tr>
<td>0367</td>
<td>C9</td>
<td>424</td>
<td>RET</td>
<td></td>
</tr>
</tbody>
</table>

; Set the status byte
; Set the status bit
; An indicator to NDOS
; Restore the address of the message
; Set up the status bit
; Do the required adjustments for hardware
; As a flag

$EJECT
LOC OBJ LINE SOURCE STATEMENT

429 ;================================================================
430 ;
431 ;
432 ;
433 ;================================================================
434 ;
435 ;================================================================
436 ;
437 ;
438 ;
439 ;================================================================
440 ;
441 ; NAME : NTWRKERROR
442 ; INPUTS : None
443 ; OUTPUTS : None
444 ; CALLS : None
445 ; DESTROYS : None
446 ; DESCRIPTION : NTWRKERROR does nothing at this stage, but is called every time a network error occurs.
447 ;
448 ;
449 ;================================================================
500 NTWRKERROR:
501 ; perform any required device re-initialization
502 RET
503
504 $EJECT
NTWRKWBOOT

NAME : NTWRKWBOOT
INPUTS : None
OUTPUTS : None
CALLS : BDOS
DESTROYS : A, C, Flags
DESCRIPTION : NTWRKWBOOT is another routine which is empty at this stage apart from the boot up message.

This procedure is called each time the CCP is reloaded from disk. At this point you could possibly check for any mail at the server or do any other task.

NTWRKWBOOT:

MVI C,9
LXI D,WBOOTMSG
CALL BDOS
RET
$EJECT
481 ;================================================================================
482 ;
483 ;  DELAY ROUTINE
484 ;================================================================================
485 ;
486 ;
487 ; NAME : DLAY
488 ; INPUTS : None
489 ; OUTPUTS : None
490 ; CALLS : None
491 ; DESTROYS : None
492 ; DESCRIPTION : DLAY creates a delay related to DLCOUNT
493 ;
494 ;================================================================================
495 DLAY:

Ô372 F5 496 PUSH PSW ; Save all the registers
Ô373 C5 497 PUSH B
Ô374 ÔECC 498 MVI C,DLCOUNT ; Create the correct time delay
Ô376 E3 499 DLAY1:
Ô377 E3 500 XTHL ; To create some type of delay
Ô378 E3 501 XTHL
Ô379 E3 502 XTHL
Ô37A ÔD 503 XTHL
Ô37B Ô276Ô3 504 DCR C ; End of the time?
Ô37C Ô1 505 JNZ DLAY1
Ô37D F1 506
Ô37E Ô 507 POP B ; Restore the Registers
Ô380 C9 508 POP PSW
Ô381 C9 510
NETWORK STATUS ROUTINE

; NAME : NTWRKSTS
; INPUTS : None
; OUTPUTS : Network status to memory NETWORKSTATUS
; CALLS : None
; DESTROYS : B, Flags
; DESCRIPTION : NTWRKSTS reads the old network status and then adjusts it to include the new status. This update occurs at regular intervals.

NTWRKSTS:
LDA NETWORKSTATUS
MOV B,A
ANI NOT (RCVERR+SNDERR)
STA NETWORKSTATUS
MOV A,B
RET

$EJECT
configuration table address

NAME : CNFGTBLADR
INPUTS : None
OUTPUTS : HL - containing the configuration table address
CALLS : None
DESTROYS : H, L
DESCRIPTION : CNFGTBLADR simply loads the address of the requesters configuration table into the HL register pair.

CNFGTBLADR:

LXI H, CNFGTBL

END

0 error(s) detected

User symbols
<table>
<thead>
<tr>
<th>LOC</th>
<th>OBJ</th>
<th>LINE</th>
<th>SOURCE STATEMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACTIVE 0010 A</td>
<td>BDOS 0005 A</td>
<td>CNFGTB 038C A</td>
<td>CONFIG 0015 A</td>
</tr>
<tr>
<td>FALSE 0000 A</td>
<td>LF 000A A</td>
<td>LOGIN 0040 A</td>
<td>MA 007F A</td>
</tr>
<tr>
<td>SLAVEI 0283 A</td>
<td>SLVID 0016 A</td>
<td>SLVID1 003E A</td>
<td>SML10 0289 A</td>
</tr>
<tr>
<td>TRUE FFFF A</td>
<td>TXS1 0310 A</td>
<td>TXSUC 0305 A</td>
<td>WBOOTM 0284 A</td>
</tr>
<tr>
<td>TAC SLAVE NETWORK I/O SYSTEM</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
APPENDIX G

The PDP 11/23 LAN Card Circuit Diagrams
PDP11/23 LAN Card

Sheet 5 of 5

Designed: QPM

September 85

Drawn: QPM
The PAL equations for the PDP LAN card are given below. In PAL 4 the := means that it is a clocked output, clocked on the input specified in brackets. As before a / means negative logic.

**PAL 1**

\[\begin{align*}
\text{IN1} &= \text{BDIN} \cdot \text{INTROE} \cdot \text{WDDAT1} + \\
&\quad \text{BDIN} \cdot \text{INTROE} \cdot \text{CNTCLK1} + \\
&\quad \text{WDDAT1} \cdot \text{CNTCLK1} \cdot \text{INTROE} \\
\text{IN2} &= \text{BDIN} \cdot \text{INTROE} \cdot \text{WDDAT1} + \\
&\quad \text{BDIN} \cdot \text{INTROE} \cdot \text{CNTCLK1} + \\
&\quad \text{WDDAT1} \cdot \text{CNTCLK1} \cdot \text{INTROE} \\
\text{A} &= \text{/BDOUT} \cdot \text{/CNTCS1} \\
\text{RE} &= \text{/BDIN} \cdot \text{/WDDAT1} \\
\text{WE} &= \text{/BDOUT} \cdot \text{/WDDAT1} \\
\text{CLK123} &= \text{/CNTCLK1} \cdot \text{/BDIN} \cdot \text{READ} + \\
&\quad \text{/CNTCLK1} \cdot \text{/BDOUT} \cdot \text{/READ} + \\
&\quad \text{/CNTCS1} \cdot \text{/BDOUT} \\
\text{CNTCS} &= \text{CS} \cdot \text{/DAL1} \cdot \text{/DAL2} \\
\text{CNTCLK} &= \text{CS} \cdot \text{DAL1} \cdot \text{DAL2} \\
\text{WDADR} &= \text{CS} \cdot \text{/DAL1} \cdot \text{DAL2} \\
\text{WDDAT} &= \text{CS} \cdot \text{DAL1} \cdot \text{DAL2} \cdot \text{/TACS} \\
\end{align*}\]

**PAL 2**

\[\text{DL0} = \text{/TACS} + \text{/T8}\]
DMAWE = /T8 * /DRQ01

DMARE = /T8 * T4 * /DRQ1

ADREN = /IDLE * TACS * T6

CLK175 = /C16M * IDLE

READY = /HOSTOK

HOSTOK = /HOSTS + TACS + IDLE

DACK = TACS * /IDLE * /T8

IDLE = /TACS * /HOSTS

TACRQ = DRQ1 * DRQ0

PAL 3

HOSTRQ = CNTCLK1 * WDDAT1

WDDOUT = /WDDAT1 * /BDIN + /DMAWE

SELWE = /DMAWE * /CSEN * /T7

SELRE = /DMARE * /CSEN

RAMWE = /CNTCLK1 * /CSEN * /BDOUT

RAMOE = /CNTCLK1 * /CSEN * /BDIN

RAMAE = /CNTCLK1 * /BDIN + /CNTCLK1 * /BDOUT

RPLCLR = WDDAT1 * CNTCLK1 + BDIN * BDOUT
RPLIN = BDIN \* BDOUT

BRPLY = QF \* /READY + /INTROE + /BDOUT \* /CNTCS1

PAL 4

WDDOC := WDOC1 (C16M)

WDOC1 := /WDDAT1 \* /BDOUT + /DMARE (C16M)

WDDG = DMARE \* WDDAT1 \* WDDOC + DMARE \* BDOUT \* WDDOC

REGSEL = /WDADR1 \* /BDOUT

DONE = /HOSTRQ \* HOSTS + DL5 + /RESET

PAL 5

INTROE = /BIAKI \* /BIAK1 \* INT5 \* INT6 \* INT7

BIAKO = /BIAKI \* /INT5 + /BIAKI \* /INT6 + /BIAKI \* /INT7

CSEN = /CSEN1 + /DACK

DIO = /RAMOE + /RE
Q-Bus Timing Diagrams

Q-Bus Read Timing

Q-Bus Write Timing

Q-Bus Interrupt Timing

(Note: Negative Logic)
APPENDIX H

The PDP LAN Card Test Software
The purpose of this program is to test the hardware constructed for the PDP 11/23. The user is given the choice of receiving the message or sending it to a specified node. The message is then displayed on the console. Messages received from the network are automatically displayed on the console.

The registers are as follows:

1. **RAMADDR** - This is the address counter for a ram read or write. The top bit gives the type of function that will follow. If A15 is set a Write operation will be done, if A15 is reset then a Read follows.

2. **RAMDATA** - Addressing this byte enables either a Read from or Write to the RAM. The function chosen is determined by the status of A15.
3. WDREGADDR - Addressing this address sets up the register address on the WD2840. These registers are not addressed directly to save IO address space.

4. WDREGDATA - This is the address used to read or write to or from the WD2840 registers.

The Interrupt on the WD2840 causes an interrupt to address 200 Octal.

The Line Time clock (LTC) causes an interrupt to 100 Octal.
Equates

TRUE = 1
FALSE = 0

; Ascii equates

NUL = 0
EOT = 32
ESC = 33
ENQ = 5
ETX = 32
BS = 10
SPC = 40
CNTRLZ = 32
CR = 15
LF = 12
DEL = 177
;-----------------------------------------------
    ;    ; Console Equates
    ;
    RCSR1 = 177560
    RBUF1 = 177562
    XCSR1 = 177564
    XBUF1 = 177566

RECCHAR = 1
`SA = 001 ; Source address for the WD board
FRAMELEN = 320. ; The frame length for the WD
BSIZE = <FRAMELEN/64>-1 ; Buffer size
RXBASE = 400+<2*FRAMELEN> ; Base of receiver buffers
TIMERND = 30. ; The network dead timer length (250 mS)
ACKRESP = 16. ; Response time for ACK (500 uS)
CBLOCK = 0 ; Position of the Control block
MYID = SA ; My Address
CYSKIP = 0 ; Number of cycles skipped at start up
MAXFRAMES = 5 ; Max number of frames transmitted per token
TXFCB = B 00000000 ; Frame control byte
WTXFCB = B 10000000 ; Frame control byte - wait for ACK
SCANFCB = B 00100000 ; Scan FCB - transparent
BASE = 177140 ;
RAMADDR = BASE + 00 ; The hardware counter
RAMDATA = BASE + 02 ; Data register
WDRBGADDR = BASE + 04 ; The address latch
WDRBGDATA = BASE + 06 ; WD2840 Status Registers`
<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>108</td>
<td>000000</td>
<td>CR0 = 0, Control Register 0</td>
</tr>
<tr>
<td>109</td>
<td>000001</td>
<td>CR1 = 1, Control Register 1</td>
</tr>
<tr>
<td>110</td>
<td>000002</td>
<td>SR0 = 2, Status Register 0</td>
</tr>
<tr>
<td>111</td>
<td>000003</td>
<td>IR0 = 3, Interrupt Register</td>
</tr>
<tr>
<td>112</td>
<td>000004</td>
<td>SR1 = 4, Status Register 1</td>
</tr>
<tr>
<td>113</td>
<td>000005</td>
<td>SR2 = 5, Status Register 2</td>
</tr>
<tr>
<td>114</td>
<td>000006</td>
<td>CTR0 = 6, Counter Register 0</td>
</tr>
<tr>
<td>115</td>
<td>000007</td>
<td>NA = 7, Next Address Register</td>
</tr>
<tr>
<td>116</td>
<td>000010</td>
<td>TA = 10, ACK timer Register</td>
</tr>
<tr>
<td>117</td>
<td>000011</td>
<td>TD = 11, Network Dead Timer</td>
</tr>
<tr>
<td>118</td>
<td>000012</td>
<td>CBPH = 12, Control Block Pointer (MSB)</td>
</tr>
<tr>
<td>119</td>
<td>000013</td>
<td>CBPL = 13, Control Block Pointer (LSB)</td>
</tr>
<tr>
<td>120</td>
<td>000014</td>
<td>NAR = 14, Next Address Request Register</td>
</tr>
<tr>
<td>121</td>
<td>000015</td>
<td>AHOLT = 15, Initial Hold-off</td>
</tr>
<tr>
<td>122</td>
<td>000016</td>
<td>TXLT = 16, Max Number of frames per Token</td>
</tr>
<tr>
<td>123</td>
<td>000017</td>
<td>MA = 17, My Address Register</td>
</tr>
<tr>
<td>124</td>
<td>000018</td>
<td></td>
</tr>
</tbody>
</table>

; Bits in the Interrupt Register (IR0)

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>000001</td>
<td>ITD = B 00000001, Network dead bit</td>
</tr>
<tr>
<td>129</td>
<td>000002</td>
<td>ITA = B 00000010, Acknowledge timeout</td>
</tr>
<tr>
<td>130</td>
<td>000004</td>
<td>ITOK = B 00000100, Token received</td>
</tr>
<tr>
<td>131</td>
<td>000010</td>
<td>IREC = B 00001000, Data frame received</td>
</tr>
<tr>
<td>132</td>
<td>000020</td>
<td>ITRAN = B 00010000, Data frame transmitted</td>
</tr>
<tr>
<td>133</td>
<td>000040</td>
<td>INS = B 00100000, New successor found</td>
</tr>
<tr>
<td>134</td>
<td>000100</td>
<td>IRO = B 01000000, Receiver over run error</td>
</tr>
<tr>
<td>135</td>
<td>000200</td>
<td>ITERM = B 10000000, Transmitter error</td>
</tr>
</tbody>
</table>
IEEE Equate

\[ \text{UI} = \text{B 00000011} \]; Unnumbered Information frame
Main program

Set the Origin

\texttt{.PSECT \_=+.500 \; ; \text{Set the origin at 500 octal}}

Main program

\texttt{\textbf{MAIN:}}
\texttt{\textbf{MOV} \#STACK,SP \; ; \text{Set up the stack pointer}}
\texttt{\textbf{JSR} PC,INITNET \; ; \text{Initialise the network}}

ML10:
\texttt{\textbf{MOV} \#MSG7,R1 \; ; \text{Ask the user what to do}}
\texttt{\textbf{JSR} PC,MSGOUT}

ML12:
\texttt{\textbf{BITB} @RCSP1,#B 100000 \; ; \text{Is there a character from the console?}}
\texttt{\textbf{BNE} ML14 \; ; \text{Yes - test to see the users requirements}}
\texttt{\textbf{BITB} IRDATA,#IREC \; ; \text{If the RXDATAFRAME flag is}}
\texttt{\textbf{BNE} ML13 \; ; \text{A receive error?}}
\texttt{\textbf{BITB} IRDATA,#IROR \; ; \text{A receive error?}}
173 MOV READLAST,R0 ;
174 000550 016700 005554 ADD #FRAMELEN,R0 ; Point at the new buffer
175 000554 062700 000500 CMP TOPADDR,R0 ; Is the pointer at the top of memory?
176 000560 026700 005546 BPL ML121 ; No - then continue
177 000564 100002 MOV #RXBASE,R0 ; Else - set R0 to the base of memory
178 000566 012700 001600 ML121: MOV #FRAMELEN,R0 ;
179 000572 000554 062700 000500 CMP TOPADDR,R0 ;
180 000576 000000 000560 000564 ADD R0,#FRAMELEN ;
181 000580 000000 000000 BIS #FRAMELEN,R0 ; Indicate that a read operation follows
182 000584 010037 177140 INC R0 ;
183 000588 113701 177142 MOV R0,#RAMADDR ;
184 000590 130127 000200 MOVB @RAMDATA,Rl ; Read the data at FSB
185 000594 001741 000000 MOV R0,0 ;
186 000598 000000 000000 ML13: JSR PC,READMSG ; set then get the frame(s)
187 00059C 004767 000074 BR ML10 ;
188 0005A0 000000 000000 ML14: MOV #1,RBUF ; Wait for a single letter response
189 0005A4 004767 000256 MOVB #CR,R4 ; Terminating character
190 0005A8 112767 000001 005502 JSR PC,CIN ;
191 0005AC 112704 000015 MOVB #CR,R4 ;
192 0005B0 004767 000256 JSR PC,READMSG ;
193 0005B4 116701 000002 MOVB RBUF,Rl ; Read the response
194 0005B8 142701 000200 FICB #B 1000000,Rl ; Clear the top bit
195 0005BC 122701 000124 CMPB #T,Rl ; T - output
196 0005C0 001403 000000 BEQ ML15 ;
197 0005C4 122701 000164 CMPB #T,Rl ;
198 0005C8 001003 000000 BNE ML20 ;
199 0005CC 004767 000132 ML15: JSR PC,SNDMSG ;
200 0005D0 000706 BR ML10 ;
ML20:
202 000674
203 000674 122701 000122
204 000700 001403
205 000702 122701 000162
206 000706 001300
207 000710

CMPB #R,Rl ; R - input
BEQ ML30 ;
CMPB #r,Rl ;
BNE ML10 ;
ML30:
208 000710 004767 000002
209 000714 000675

JSR PC,RECM SG ;
BR ML10 ;
Message from the network

NAME : RECMSG
INPUTS : Message from the network
OUTPUTS : Memory INBUF - containing the message received
CALLS : MSGOUT, IICSA
DESTROYS : R1, R2, Flags
DESCRIPTION : Reads the message in from the network adjusting the Source address into ASCII and then prints the message.

;------------------------------------------------------------
; NAME : RECMSG
; INPUTS : Message from the network
; OUTPUTS : Memory INBUF - containing the message received
; CALLS : MSGOUT, IICSA
; DESTROYS : R1, R2, Flags
; DESCRIPTION : Reads the message in from the network adjusting the Source address into ASCII and then prints the message.
;------------------------------------------------------------

RECMSG:
MOVB #MSGB,R1          ; Enable the user to leave
JSR  PC,MSGOUT         ;

RML10:
BITB @#RCSR!,#B 10000000 ; See if there is a character
BNE  RMEXIT            ;

RML25:
BICB #IREC,IRDATA      ; Clear the flags
BICB #IRED,IRDATA      ;

; Now go and receive the message

RML30:
JSR  PC,READWDBUF      ;
CMPB #TRUE,NODATA      ; Was data read?
BEQ  RMEXIT           ; No - then exit
JSR  PC,IICSA          ; Reverse the ASCII procedure
MOV #MSG9,R1 ; Write out the source code
JSR PC,MSGOUT ;
MOV #INBUF+1,R1 ; Write out the message received
JSR PC,MSGOUT ;
CMPB #TRUE,NOMOREDATA ; Any more data?
BNE RML30 ; Loop if there is more data

RMEXIT:
RTS PC ;
Send the message over the network

**NAME**: SNDMSG

**INPUTS**: None

**OUTPUTS**: Message over the network

**CALLS**: MSGOUT, CIN, UNASCII, FILTXBUF

**DESTROYS**: Rl, Flags

**DESCRIPTION**: Prompts the user for the destination node, and then the user is asked to input the message to be transmitted. This procedure will then instruct FILTXBUF to write the message received to the next buffer for transmission.

```
268 001024
269 001024 012701 004756' MOV #MSG5,Rl ; Prompt the user for details
270 001030 004767 000520 JSR PC,MSGOUT;
271 001034 112767 000003 005272 MOVB #3,RBUF ; Set up the max characters expected
272 001042 112704 000015 MOVB #CR,R4 ; Terminating character
273 001046 004767 000046 JSR PC,CIN ; Wait for the response from the user
274 001052 004767 000220 JSR PC,UNASCII ; Convert the ascii destination to a hex character
275
276 001056 012701 005203' MOV #MSG6,Rl ; Ask for the message now
277 001062 004767 000466 JSR PC,MSGOUT;
278 001066 012767 001212 005240 MOV #650.,RBUF ; Accept a frame of max length 650 bytes
279 001074 112704 000032 MOVB #CNTRL2,R4 ; Terminating character
280 001100 004767 000014 JSR PC,CIN ;
281 001104 112767 000003 005206 MOVB #UI,CONTTYPE ; Type of IEEE control byte
282 001112 004767 001526 JSR PC,FILTXBUF ; Fill in the network buffer
283 001116 000207 RTS PC ;
```
Console Input routine

; NAME: CIN
; INPUTS: R4: - Termination character
; Memory MAXNO - containing the max number of characters allowed.
; OUTPUTS: Memory NOENT - containing the number of characters read
; BUF - containing the characters read in from the console.
; CALLS: MSGOUT
; DESTROYS: Flags
; DESCRIPTION: Reads in the characters from the console comparing each
; with CR and DEL. A CR will terminate the input, after
; which the number of characters read in from the console
; is entered into the NOENT field. A DEL character will
; cause the message <BS>,<SP>,<BS> to be sent out. If the
; number of characters input is equal to MAXNO the input
; is terminated.

CIN:

MOV R2,-(SP) ; Save all the registers which are to
MOV R1,-(SP) ; used
MOV R0,-(SP) ;
MOV #RBUPI,R1 ; Set up the pointer to the input buffer
CLR R2 ; Zero the counter
; To deal with the special case of the DEL

CIDEL:

MOV R1,-(SP) ; Save current status
MOV #DELMSG,R1 ;
JSR PC,MSGOUT ; Send BS,SP,BS to console
MOV (SP)+,R1 ; Restore the data pointer
DEC R2 ; Adjust the counter
BR CIL05 ; Wait for a new character

; Point of exit

CEXIT:

MOV R2,RBUFD ; Write the counter to memory
MOVB #NUL,(R1) ; Sign off the message in the buffer
MOV (SP)+,R0 ; Restore the registers used to
MOV (SP)+,R1 ; their original value
MOV (SP)+,R2 ;
RTS PC ;
Delay routine

; NAME : DELAY
; INPUTS : None
; OUTPUTS : None
; CALLS : None
; DESTROYS : Flags
; DESCRIPTION : Delays the program for a short time.

DELAY:

   MOV  R0,-(SP)  ;
   MOV  #10000,R0  ; Counter

DL10:

   SOB  R0,DL10  ;
   MOV  (SP)+,R0  ; Restore register
   RTS  PC  ;
<table>
<thead>
<tr>
<th>NAME</th>
<th>INPUTS</th>
<th>OUTPUTS</th>
<th>DESTROYS</th>
<th>CALLS</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>UNASCII</td>
<td>Memory - RBUF containing up to two ASCII characters</td>
<td>R2,-(SP)</td>
<td>DESTADDR</td>
<td>_</td>
<td>Converts the ASCII digits read in via CNV into a destination address.</td>
</tr>
<tr>
<td>INPUTS</td>
<td>Memory - DESTADDR containing the destination address</td>
<td>R1,-(SP)</td>
<td>_</td>
<td>_</td>
<td>Read in the number of characters read.</td>
</tr>
<tr>
<td>CALLS</td>
<td>DOASCII</td>
<td>RBUFD,RL</td>
<td>_</td>
<td>_</td>
<td>Save all the registers which are to be used.</td>
</tr>
<tr>
<td>DESTROY</td>
<td>UNASCII:</td>
<td>MOV</td>
<td>MOV</td>
<td>JSR</td>
<td>UNASCII:</td>
</tr>
<tr>
<td>VALID</td>
<td>UA10:</td>
<td>MOV</td>
<td>MOV</td>
<td>JSR</td>
<td>UA10:</td>
</tr>
<tr>
<td>IF two digits</td>
<td>MOV</td>
<td>MOV</td>
<td>JSR</td>
<td>UA10:</td>
<td></td>
</tr>
</tbody>
</table>

```asm
; Untangle the ASCII characters
; Save all the registers which are to be used
; Read in the number of characters read
; Do the ASCII conversion

MOV R2, - (SP)    ; If two digits
MOV R1, - (SP)    ; Save all the registers which are to be used
RBUFD, RL        ; Read in the number of characters read
PC, DOASCIICON    ; Do the ASCII conversion
```

- Unscramble the ASCII characters
- Save all the registers which are to be used
- Read in the number of characters read
- Do the ASCII conversion
MOV R1,R2 ; Read in the next character
MOV RBUFI+1,R1
BR UA40

; If only a single digit was read in

UA30:
MOV #NUL,R2 ; Set up the first digit as zero
MOV RBUFI,R1

; Now both possibilities converge

UA40:
JSR PC,DOASCICON ;
BICB $B 11110000,R1 ; Clear the top 4 bits
XOR R1,R2 ; Combine the two nibbles
CMPB &0.,R2 ;
BEQ UAERROR ; If 0 then error.
MOV R2,DESTADDR ; Save result

; All done

UAEXIT:
MOV (SP)+,R1 ; Restore the registers used to
MOV (SP)+,R2 ; their original value
RTS PC ;
; Error in destination address

UAERROR:
    MOV    #MSG65,R1 ; Tell the user of his mistake
    JSR    PC,MSGOUT

        MOVB   #3,RBUF ; Read in the new address
        MOVB   #CR,R4 ; Terminating character

        BR    UA10 ; Try again

DOASCIICON:
    BITB   R1,#B 01000000 ; Is bit six set (i.e. A-F)
    BEQ    DACEXIT

        ADD    #9.,R1 ; Adjust A-F

DACEXIT:
    BICB   #B 11110000,R1 ; Clear the top nibble and return
    RTS    PC ;
Conversion routine for the SA

NAME: IICSA

INPUTS: Memory INBUF - containing Source address in HEX

OUTPUTS: Memory MSG9I - containing ASCII Source Address

CALLS: None

DESTROYS: R1, Flags

DESCRIPTION: Simply converts the SA from HEX to ASCII code.

IICSA:

MOV B INBUF,R1 ; Read SA
MOV B R1,R2
ASR R1 ; MSB first
ASR R1
ASR R1
ASR R1
BIC B #B 11110000,R1 ; Clear the top nibble
CMP B R1,#B 00001010 ;
BMI IIL10 ;
ADD #7,R1 ;

IIL10:

ADD #B 00110000,R1 ; For an ascii digit
MOV B R1,MSG9I ; Insert value in msg

MOV B R2,R1 ;

BIC B #B 11110000,R1 ;
CMP B R1,#B 00001010 ;
<table>
<thead>
<tr>
<th>Line</th>
<th>Hex</th>
<th>Binary</th>
<th>Operation</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>502</td>
<td>001534</td>
<td>100402</td>
<td>BMI</td>
<td>IIL20</td>
</tr>
<tr>
<td>503</td>
<td>001536</td>
<td>062701</td>
<td>ADD $7.,R1</td>
<td></td>
</tr>
<tr>
<td>504</td>
<td>001542</td>
<td></td>
<td>IIL20:</td>
<td></td>
</tr>
<tr>
<td>505</td>
<td>001542</td>
<td>062701</td>
<td>ADD $B 00110000,R1</td>
<td></td>
</tr>
<tr>
<td>506</td>
<td>001546</td>
<td>110167</td>
<td>MOVB R1,MSG9I+1</td>
<td></td>
</tr>
<tr>
<td>507</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>508</td>
<td>001552</td>
<td>000207</td>
<td>RTS PC</td>
<td></td>
</tr>
</tbody>
</table>
Console Message out routine

; NAME : MSGOUT
; INPUTS : R1 - pointing at message
; OUTPUTS : Message to the console
; CALLS : None
; DESTROYS : R1, Flags
; DESCRIPTION : Writes the message to the console, checking for a NUL which indicates the end of message.

MSGOUT:

BITB @#XCSR1,#B 10000000
BEQ MSGOUT
CMPB (R1),#NUL
BEQ MOEXIT
MOVB (R1)+,@#XBUF1
BR MSGOUT

MOEXIT:

RTS PC
MAC Layer Routines

1. WD2840 Initialisation

NAME : INITNET
INPUTS : None
OUTPUTS : None
CALLS : MSGOUT
DESTROYS : R1, R2, Flags
DESCRIPTION : INITNET sets up the parameters as required by the WD 2840 and informs the user of the progress. If the network is dead then the LAN controller will seek to start it up. If the network is alive then the Controller will check to see if there is a duplicate address if so it will inform the user and stop. Otherwise it will enter the ring on the next token pass.

INITNET:

; Welcome the user

; Write the message to the terminal.

MOV #MSG1,R1
JSR PC,MSGOUT
; Set up the control block

566
567
568 001612 012737 077777 171140
569 001620 012701 001600
570 001624 000301
571 001626 110137 177142
572 001632 000301
573 001634 110137 177142
574
575 001640 012701 000400
576 001644 000301
577 001646 110137 177142
578 001652 000301
579 001654 110137 177142
580 001660 112737 000004 177142
581
582 001666 012701 000013
583 001672 112737 000000 177142
584 001700 005301
585 001702 001373
586
587
588
589 001704 012700 001577
590 001710 010001
591 001712 005201
592 001714
593 001714 062701 000500
594 001720 030127 020000
595 001724 001020
596 001726 042700 100000

; Write to address 0
; Write out MSB of address first
; Now LSB
;
; Then the transmit address
; The MSB <> 0 thus 400
;
; Set up the buffer size
;
; Zero the following 11 places
for event counters
;
; Find the initial buffer
;
; Get the next address
;
; Indicate that a write operation
MOV R0,#@RAMADDR  
; follows

SWAB R1  
; Write the MSB first

MOV R1,#@RAMDATA  

MOV R1,#@RAMDATA  
; Zero the FSB

ADD #FRAMELEN,R0  
; Adjust the pointer

MOV R0,#@RAMADDR  
; Zero the last link

MOV R0,#@RAMADDR  

MOV #12J,#@RAMDATA  

MOV #0,#@RAMDATA  

MOV #12J,#@RAMDATA  
; And the FSB

MOV #RXBASE-FRAMELEN,READLAST  
; Set up the input buffer pointer

MOV #2,TXBUF  
; Second buffer used

MOV #TA,R5  
; Save the present WD reg addressed

MOV R5,#@WRGADDR  

MOV #ACKRESP,#@WRGDATA  
; Set up ACK timer

INC R5  
; R5 points at TD register
629 002070 112737 000036 177146  MOV #TIERND, @WDRGDATA ; Set up network dead timer
630 002076 012701 000000  MOV #CBLOCK, R1
631 002102 012705 000013  MOV #CBPL, R5
632 002106 110537 177144  MOV R5, @WDRGADDR ; Set up the address of register
633 002112 110137 177146  MOV R1, @WDRGDATA ; Write out the control block address
634 002116 000301  SWAB R1
635 002120 012705 000012  MOV #CBPH, R5
636 002124 110537 177144  MOV R5, @WDRGADDR ; Set up the address of register
637 002130 110137 177146  MOV R1, @WDRGDATA
638 002134 012705 000014  MOV #NAR, R5
639 002140 110537 177144  MOV R5, @WDRGADDR
640 002144 112737 000002 177146  MOV #MYID+1, @WDRGDATA ; Next Address set up
641 002152 005205  INC R5 ; R5 points at AHOLT
642 002154 110537 177144  MOV R5, @WDRGADDR ;
643 002160 12737 000000 177146  MOV #CYSKIP, @WDRGDATA ; Miss cycles
644 002166 005205  INC R5 ; Points at TXLT
645 002170 110537 177144  MOV R5, @WDRGADDR ;
646 002174 112737 000005 177146  MOV #MAXFRAMES, @WDRGDATA ; Set the max frames
647 002202 005205  INC R5 ; Points at MA
648 002204 110537 177144  MOV R5, @WDRGADDR ; Set up the address of register
649 002210 112737 000001 177146  MOV #MYID, @WDRGDATA ; Set up my address
650
651 ; Do a dummy start to see if the network is dead
652
653 002216 012705 000001  MOV #CR1, R5
654 002222 110537 177144  MOV R5, @WDRGADDR ;
655 002226 112737 000000 177146  MOV #0., @WDRGDATA ; Zero the Control regs
656 002234 012705 000000  MOV #CR0, R5
657 002240 110537 177144  MOV R5, @WDRGADDR ;
658 002244 112737 000000 177146  MOV #0., @WDRGDATA ;
; Wait for confirmation
; Let the WD update its regs
; Wait for SR21 to clear confirming
; a state change
; Reset the timer counter
; Clear the interrupt register save
; space
; Let the WD update its regs
; Has there been a NETDEAD interrupt?
; Wait for 300 msec
; The program reaches here if the network is NOT dead
; so enter into the swing of things by claiming the next token

; Enter scan mode, take new NA
; Enable token interrupt and Control frame transmission
; Watch for address duplication
688
689
MOV #MSG10,RL

; Tell user of progress
690 002372 012701 004442
691 002376 004767 177152
692
693 002402 142767 000004 003715
BICB #ITOK,IRDATA

; Clear the TOKEN RECEIVED bit in the
interrupt storage space
694
695 002410 012767 000000 003704
MOV #0.,TIMERCOUNT

; Reset the timer counter
696 002416 ADL10:
697 002416 012701 000002
MOV #2.,RL

; Token counter
698 002422 136727 003677 000004
BITB IRDATA,#ITOK

; Was there a token received?
699 002430 001002
BNE ADL20

; If token then decrement RL
700 002432 005301
DEC RL

701 002434 001411
B EQ ADL15

702 002436 ADL20:
703 002436 026727 003660 000062
CMP TIMERCOUNT,#50.

; Wait for 1 sec
704 002444 101364
BHI ADL10

705
706 002446 012701 004577
MOV #MSG2,RL

; Tell user of progress
707 002452 004767 177076
JSR PC,MSGOUT

708
709 002456 000411
BR NDL05

; Now get into the loop
710
711 002460 ADL15:
712 002460 012701 004650
MOV #MSG3,RL

; Die as nothing else can be done
713 002464 004767 177064
JSR PC,MSGOUT

714 002470 000000
HALT
; Get into the logical loop

NETDEAD:

MOV #MSG11,R1 ; Tell user of progress
JSR PC,MSGOUT

NDL05:

MOV %CR1,R5 ; Enables the TAC to enter the loop
MOV B 01101001,#WDRIDATAD ; Enables the TAC to enter the loop
MOV %CR0,R5
MOV B 01100001,@WDREXDATA ; Enable token interrupt and control information frames

NDL10:

JSR PC,DELAY ; Let the TAC update
MOV #SR2,R5 ; Only concerned with lowest two bits
BICB #B 11111100,R1 ; Is STATE clear, and INRING set?
CMPB R1,#B 000000001 ; Loop until ready

; Enable the transmission of frames

MOV %CR0,R5 ; Disable token receive interrupts
MOV R5,#WDRIDATAD
MOV B 01100000,@WDREXDATA
MOV R5,#WDRIDATAD

747 002616 113701 177146 MOVB @WREGDATA,R1 ; Read from Crl
748 002622 152701 000100 BISB #B 01000000,R1 ;
749 002626 110137 177146 MOVB R1,#WREGDATA ; Read the status ;
750 ;
751 002632 012701 004722 MOV #MSG4,R1 ;
752 002636 004767 176712 JSR PC,MSGOUT ;
753 ;
754 002642 000207 RTS, PC ; Ready for transmission
;----------------------------------------------------------
; 2. Fill in the Network transmit buffer
;----------------------------------------------------------
; NAME        : FILTXBUF
; INPUTS      : Memory DESTADDR - containing the destination address
;              RBUPI - containing the message to be sent
; OUTPUTS     : Message to the network
; CALLS       : None
; DESTROYS    : R0, R1, R2, Flags
; DESCRIPTION : Finds the next available buffer and then fills in the
;               data from the CIN buffer.
;----------------------------------------------------------

FILTXBUF:
MOV   #RBUFI,R4 ; Set up the pointer to the data

; Find out which buffer is to be used

FTXL0:
CMPB  #1,TXBUF  ; First or second buffer used
BEQ   FTXL01    ;
MOV   #400,R1    ; - Used second last time, so first this time
MOVB  #1,TXBUF   ;
BR    FTXL05    ;

FTXL01:
MOV   #400+FRAMELEN,R1 ; - Use second this time
MOVB  #2,TXBUF   ;
DEC Rl ; For the hardware
BIC #100000,R1 ; Write operation
MOV R1,#RAMADDR ;
MOVB #0,#RAMDATA ; Clear links and FSB
MOVB #0,#RAMDATA ;
CMPB #377,DESTADDR ; Is it a broadcast frame?
BEQ FTXL10 ;
MOVB #WXPCB,#RAMDATA ; Set up the control byte (Wait for ACK)
BR FTXL20 ;
FTXL05:
FTXL06:
FTXL07:
FTXL08:
FTXL09:
FTXL10:
FTXL11:
FTXL12:
FTXL13:
FTXL14:
FTXL15:
FTXL16:
FTXL17:
FTXL18:
FTXL19:
FTXL20:
FTXL21:
FTXL22:
FTXL23:
FTXL24:
FTXL25:
FTXL26:
FTXL27:
FTXL28:
FTXL29:
FTXL30:
FTXL31:
FTXL32:
FTXL33:
FTXL34:
FTXL35:
FTXL36:
FTXL37:
FTXL38:
FTXL39:
SWAB R3 ; MSB of length first
MOVB R3,@#RAMDATA ;
SWAB R3 ; Then LSB of length
MOVB R3,@#RAMDATA ;
MOVB DESTADDR,@#RAMDATA ; Then DA
MOVB #SA,@#RAMDATA ; and SA
MOVB CONTYPE,@#RAMDATA ; and the IEEE control byte

; Now send out the data
FTXL40:

FTXL50:

FTXL55:

; Link up and wait for transmission

849 003200 005301 \ DEC R1 ; To point at first link byte
850 003202 010137 177140 \ MOV R1,@#RAMADDR;
851 003206 110237 177142 \ MOVB R2,@#RAMDATA ; Then MSB
852 003212 000302 \ SWAB R2 ; Restore R2
853
854 ; Enable transmission of data
855
856 003214 012705 000000 \ MOV #CR0,R5 ;
857 003220 110537 177144 \ MOVB R5,@#WDRBGADDR ;
858 003224 112737 000340 177146 \ MOVB #B 11100000,@#WDRBGDATA ; Enable data transmission
859
860 003232 005202 \ INC R2 ; Point at FSB
861 003234 052702 100000 \ BIS #100000,R2 ; Read operation
862 003240 \ FTXL60:
863 003240 010237 177140 \ MOV R2,@#RAMADDR ;
864 003244 113701 177142 \ MOVB @#RAMDATA,R1 ; Read FSB
865 003250 130127 000200 \ BITB R1,#B 10000000 ; DONE set?
866 003254 001002 \ BNE FTXL65 ; Yes - then continue
867 003256 000001 \ WAIT ; Wait for an interrupt
868 003260 000767 \ BR FTXL60 ; And loop
869 003262 \ FTXL65:
870 003262 130127 000007 \ BITB R1,#B 00000111 ; If any of the lower 3 bits set = error
871 003266 001006 \ BNE FTXEX ;
872 003270 122767 000000 003021 \ CMPB #FALSE,LONG ; If LONG then loop
873 003276 001402 \ BEQ FTXEX ;
874 003300 000167 177344 \ JMP FTXL0 ;
876 ; Finish up
877
878 003304
879 003304 012705 000000
880 003310 110537 177144
881 003314 112737 000140 177146
882 003322 000207

FTXEX:
MOV #CR0,R5 ;
MOVB R5,#WDREGADDR ;
MOVB #B 01100000,#WDREGDATA ; Disable transmit
RTS PC ;
3. Read the WD buffer

NAME : READWDBUF
INPUTS : Frame in WD local memory
OUTPUTS : Memory INBUF - containing the message
CALLS : None
DESTROYS : R0, RL, R2, Flags
DESCRIPTION : Reads the data in from the local memory on the WD board into main memory.

READWDBUF:
MOV READLAST,R0 ;
ADD $FRAMELEN,R0 ; Point at the new buffer
CMP TOPADDR,R0 ; Is the pointer at the top of memory?
BPL RBL01 ; No - then continue
BNE RBL05
MOV $RXBASE,R0 ; Else - set R0 to the base of memory
RBL01:
MOV R0,R4 ; As a check byte
BIS $10000000,R0 ; Indicate that a read operation follows
INC R0 ;
MOV R0,@@RAMADDR ;
MOVB @@RAMDATA,RL ; Read the data at FSB
BITB RL, #B 10000000 ; If zero then return
BNE RBL05 ;
; Check the following frame to see if received correctly
; (occurs if receiver error)

DEC R0
BIC $100000,R0

ADD #FRAMELEN,R0
CMP TOPADDR,R0
BPL RBL03
MOV $RKBASE,R0

RBL03:

MOV R0,R4
INC R0
BIS il000000,R0
MOV @iRAMADDR,R1
BITB R1,#B 10000000
BNE RBL05

MOVB iTRUE,NODATA
JMP RBE01

RBL05:

DEC R0
BIC #100000,R0
MOV R0,RF.ADFIRST
MOV R0,RF.ADLAST

; Or receive the data

; Definitely no data

MOVB #TRUE,NODATA ; If no frames received
JMP RBE01

; And the last buffer read

; Read the length word
INC R0 ;
INC R0 ; Point at the length field
BIS #100000,R0 ; Indicate that a read operation follows
MOV R0,@#RAMADDR ; Read the length word
MOVB @#RAMDATA,R1 ; Clear top byte
SWAB R1 ;
MOVB @#RAMDATA,R2 ; Clear the top byte
BIC #177400,R2 ; Combine the length
XOR R2,R1
SUB #7,R1 ; Adjust for the overhead bytes
MOV @#INBUF,R2 ; Set up the input pointer
MOV #7,R3 ; The internal buffer pointer

; Ready to read in the data
RBL07: MOVB @#RAMDATA,(R2) ; Dummy read to adjust the pointer
RBL10: MOVB @#RAMDATA,(R2)+ ; Read the data from the WD buffer
CMP #ENDBUF,R2 ; At the bottom of data area?
BNE RBL11
DEC R2 ; Adjust the pointer to prevent corruption
RBL11: DEC R1 ; End of the frame?
INC R3; End of the present buffer?

BNE RBL10; Yes - then adjust things else loop

; End of buffer sequence

DEC R1; Adjust the counter for link bytes

DEC R1;

BEQ RBL20;

ADD #FRAMELEN,R4;

CMP TOPADD,R4; Top of buffer?

BPL RBL15;

MOV RXBASE,R4; Yes - then point at bottom of buffers

MOV R4,@fl:RAMADDR; Write out the new address

BR RBL17;

RBL15:

MOV @fl:RAMDATA,(R2); Dummy read to adjust RAM pointer

RBL17:

MOV $2,R3; Reset the buffer counter

MOV R4,READLAST;

BR RBL07; Continue reading

; End of Frame sequence - Fill in the link addresses

RBL20:

MOV #NUL,(R2); Sign off the message

MOV READLAST,R0; Find the first buffer read
```
1009 003702 010001 MOV  R0, R1       ;
1010 003704 026700 002422 CMP  TOPADDR, R0 ; Link to next buffer first
1011 003710 001002 BNE  RBL21       ;
1012 003712 012701 001100 MOV  #RXBASE-FRAMELEN, R1 ;
1013 003716 062701 000500 RBL21:  ADD  #FRAMELEN, R1
1014 003716 062701 000500 BIC  #100000, R0 ; Write LSB to the higher addr first
1015 003722 012701 001100 MOV  R0, @#RAMADDR
1016 003726 042700 100000 MOV  R0, @#RAMDATA
1017 003732 110137 177142 MOVB  R1, @#RAMDATA
1018 003736 000301 SWAP  R1
1019 003740 005300 DEC  R0
1020 003742 010001 MOV  R0, R1       ; No - then continue as normal
1021 003746 010001 MOV  R0, R2
1022 003760 001600 CMP  #RXBASE, R0
1023 003764 001004 BNE  RBL22       ; Else - point at the top buffer
1024 003766 016700 002340 MOV  TOPADDR, R0
1025 003772 062700 000500 ADD  #FRAMELEN, R0
1026 003776 162700 000500 RBL22:  SUB  #FRAMELEN, R0
1027 003776 162700 000500 BIC  #100000, R0 ; Get to the link bytes
1028 003782 001004 MOV  R0, R2       ; Link the previous buffer to the present
1029 003786 010000 100000 DEC  R0
1030 003806 005300 BNE  RBL22       ; Adjust for hardware
1031 003810 010037 177140 MOV  R0, @#RAMADDR
1032 003814 112737 000000 177142 MOVB  #0, @#RAMDATA ; Zero previous buffers link
1033 003822 112737 000000 177142 MOVB  #0, @#RAMDATA
1034 003830 112737 000000 177142 MOVB  #0, @#RAMDATA
1035 003836 005200 INC  R0
1036 003836 005200 INC  R0
```
```
1040 004040 RBL25:
1041 004040 010001 MOV R0,R1
1042 004042 022700 001600 CMP #RXBASE,R0
1043 004046 001004 BNE RBL27
1044 004050 016700 002256 MOV TOPADDR,R0
1045 004054 062700 000500 ADD #FRAMELEN,R0
1046 004060 RBL27:
1047 004060 162700 000500 SUB #FRAMELEN,R0
1048 004064 042700 100000 BIC #100000,R0
1049 004070 010037 177140 MOV R0,@@RAMADDR
1050 004074 110137 177142 MOVB R1,@@RAMDATA
1051 004100 112737 000000 177142 MOVB #0,@@RAMDATA
1052 004106 000301 SWAB R1
1053 004110 005300 DEC R0
1054 004112 010037 177140 MOV R0,@@RAMADDR
1055 004116 110137 177142 MOVB R1,@@RAMDATA
1056 004122 000301 SWAB R1
1057 004124 005200 INC R0
1058
1059 004126 020267 002174 CMP R2,READFIRST
1060 004132 001402 BEQ RBL30
1061 004134 010102 MOV R1,R2
1062 004136 000740 BR RBL25
1063
; Check to see if there are any more frames to be read
1064
1065
1066 004140 RBL30:
1067 004140 112767 000000 002147 MOVB #FALSE,NODATA
1068 004146 016700 002156 MOV READLAST,R0
1069 004152 062700 000500 ADD #FRAMELEN,R0
1070 004156 026700 002150 CMP TOPADDR,R0
```
1072 004162 100002  
1073 004164 012700 001600  
1074 004170  
1075 004170 005200  
1076 004172 052700 100000  
1077 004176 010037 177140  
1078 004202 113701 177142  
1079 004206 130127 000200  
1080 004212 001023  
1081  
1082  
1083  
1084 004214 062700 000500  
1085 004220 042700 100000  
1086 004224 026700 002102  
1087 004230 100002  
1088 004232 012700 001600  
1089 004236  
1090 004236 005200  
1091 004240 052700 100000  
1092 004244 010037 177140  
1093 004250 113701 177142  
1094 004254 130127 000200  
1095 004260 001404  
1096  
1097  
1098  
1099 004262  
1100 004262 112767 000000 002024  
1101 004270 000403  

1072 004162 100002 BPL RBL35
1073 004164 012700 001600 MOV #RXBASE,R0
1074 004170 RBL35:
1075 004170 005200 INC R0
1076 004172 052700 100000 BIS #100000,R0 ; Read operation to follow
1077 004176 010037 177140 MOV R0,#RAMADDR ; See if the next buffer is full
1078 004202 113701 177142 MOVB #RAMDATA,R1
1079 004206 130127 000200 BITB R1,#B 10000000 ; Is the buffer ready?
1080 004212 001023 BNE RBEXIT
1081
1082 ; Check the following frame
1083
1084 004214 062700 000500 ADD #FRAMELEN,R0
1085 004220 042700 100000 BIC #100000,R0
1086 004224 026700 002102 CMP TOPADDR,R0 ; Top of buffers?
1087 004230 100002 BPL RBL37
1088 004232 012700 001600 MOV #RXBASE,R0
1089 004236 RBL37:
1090 004236 005200 INC R0
1091 004240 052700 100000 BIS #100000,R0 ; Read operation to follow
1092 004244 010037 177140 MOV R0,#RAMADDR ; See if the next buffer is full
1093 004250 113701 177142 MOVB #RAMDATA,R1
1094 004254 130127 000200 BITB R1,#B 10000000 ; Is the buffer ready?
1095 004260 001404 BEQ RBEXIT1
1096
1097 ; Exit point of procedure
1098
1099 004262 RBEXIT:
1100 004262 112767 000000 002024 MOVB #FALSE,NOMOREDATA ; Not the last frame
1101 004270 000403 BR RBEXIT1

B-44
1103 ; Show if there is no data in the next buffer
1104
1105 004272
1106 004272 112767 000001 002014
1107 004300
1108 004300 000207

RBE01:
MOV B #TRUE, NOMOREDATA;
RBE10:
RTS PC ;
Interrupt Service Routines

1. Line Time Clock Interrupt Vector

LTCSR:

<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>004302</td>
<td>MOV R1,-(SP)</td>
<td></td>
</tr>
<tr>
<td>004302</td>
<td>INC</td>
<td>Increment the general timer</td>
</tr>
<tr>
<td>004304</td>
<td>INC</td>
<td>and the SCAN time counter</td>
</tr>
<tr>
<td>004310</td>
<td>CMPB SCANTCOUNT,#50.</td>
<td>If 1 Sec passed then scan</td>
</tr>
<tr>
<td>004314</td>
<td>LTCEXIT</td>
<td></td>
</tr>
<tr>
<td>004322</td>
<td>LTCEXIT</td>
<td></td>
</tr>
<tr>
<td>004324</td>
<td>MOV R5,-(SP)</td>
<td>If interrupted by WDSR</td>
</tr>
<tr>
<td>004326</td>
<td>MOV #CR1,R5</td>
<td>Point at the CR1</td>
</tr>
<tr>
<td>004332</td>
<td>MOVB R5,@WDSREGADDR</td>
<td>Instruct WD to take NAR</td>
</tr>
<tr>
<td>004336</td>
<td>MOVB #151,#WDSREGDATA</td>
<td></td>
</tr>
<tr>
<td>004344</td>
<td>MOV (SP)+,R5</td>
<td></td>
</tr>
<tr>
<td>004346</td>
<td>MOVB R5,@WDSREGADDR</td>
<td>Point at the old value</td>
</tr>
<tr>
<td>004352</td>
<td>MOVB #0,SCANTCOUNT</td>
<td>Zero the counter</td>
</tr>
<tr>
<td>004360</td>
<td>LTCEXIT</td>
<td></td>
</tr>
<tr>
<td>004360</td>
<td>MOV (SP)+,R1</td>
<td></td>
</tr>
<tr>
<td>004362</td>
<td>RTI</td>
<td></td>
</tr>
</tbody>
</table>

; Set NA to MA+1 to enable new nodes to enter net
2. Western Digital LAN Controller Interrupt Service Routine

WDSR:

MOVB #IR0, @WDREGADDR ; Read the vector
BISB @WDREGDATA, IRDATA ; AND the bits set in the IR0 register with those in the storage space below.
BISB @WDREGDATA, IRDATA ; Must be done twice
MOVB RS, @WDREGADDR ; Restore pointer to WD reg
RTI
<table>
<thead>
<tr>
<th>Line</th>
<th>Message Details</th>
</tr>
</thead>
</table>
| 1156 | | ;==================================================================
| 1157 | | ;
| 1158 | | ; Messages
| 1159 | | ;
| 1160 | | ;
| 1161 004414 | MSG1: .BYTE ESC
| 1162 004414 033 | .ASCII "H"
| 1163 004415 110 | .BYTE ESC
| 1164 004416 033 | .ASCII "J"
| 1165 004417 112 | .BYTE 0
| 1166 004420 127 105 114 | .ASCII "WELCOME TO PDPLAN"
| 1167 004441 000 | .BYTE 0
| 1168 004442 | MSG10:
| 1169 004442 015 012 012 | .BYTE CR,LF,LF,LF
| 1170 004446 124 105 123 | .ASCII "TESTING FOR DUPLICATE ADDRESS"
| 1171 004503 015 012 | .BYTE CR,LF
| 1172 004505 000 | .BYTE 0
| 1173 004506 | MSG11:
| 1174 004506 015 012 012 | .BYTE CR,LF,LF,LF
| 1175 004512 116 105 124 | .ASCII "NETWORK DEAD - BEGINNING INITIALISATION OF NETWORK"
| 1176 004574 015 012 | .BYTE CR,LF
| 1177 004576 000 | .BYTE 0
| 1178 004577 | MSG2:
| 1179 004577 116 117 040 | .ASCII "NO ADDRESS DUPLICATION - ENTERING LOOP"
| 1180 004645 015 012 | .BYTE CR,LF
| 1181 004647 000 | .BYTE 0
| 1182 004650 | MSG3:
| 1183 004650 101 104 104 | .ASCII "ADDRESS DUPLICATION - CHANGE MY ADDRESS"
| 1184 004717 015 012 | .BYTE CR,LF
| 1185 004721 000 | .BYTE 0
MSG4:
.BYTE CR,LF
.ASCI "READY FOR DATA EXCHANGE"
.BYTE CR,LF

MSG5:
.BYTE CR,LF
.ASCI "TO WHICH TERMINAL MUST THE MESSAGE BE SENT?"
.BYTE CR,LF
.ASCI "01H TO FEH - TO SEND TO A SPECIFIC TERMINAL"
.BYTE CR,LF
.ASCI "FFH - TO BROADCAST"
.BYTE CR,LF
"ENTER THE HEX VALUE >"

MSG6:
.BYTE CR,LF,LF
.ASCI "NOW ENTER THE MESSAGE TO BE SENT (Z TO TERMINATE):"
.BYTE CR,LF
.BYTE 0

MSG65:
.BYTE CR,LF
.ASCI "AN ERROR OCCURED IN ENTERING THE ADDRESS, PLEASE RE-ENTER "
.ASCI "(01H-0FFH)"
.BYTE 0
1213 005406
1214 005406 015 012 012 .BYTE CR,LF,LF
1215 005411 124 131 120 .ASCII "TYPE: T TO TRANSMIT A FRAME"
1216 005444 015 012 .BYTE CR,LF
1217 005446 040 040 040 .ASCII "R TO RECEIVE A FRAME"
1218 005501 015 012 .BYTE CR,LF
1219 005503 040 040 040 .ASCII " > "
1220 005513 000 .BYTE 0
1221 005514 .MSG8:
1222 005514 015 012 .BYTE CR,LF
1223 005516 110 111 124 .ASCII "HIT ANY KEY TO EXIT"
1224 005541 000 .BYTE 0
1225 005542 .MSG9:
1226 005542 015 012 .BYTE CR,LF
1227 005544 115 105 123 .ASCII "MESSAGE RECEIVED FROM NODE"
1228 005577 060 060 110 .MSG9I: .ASCII "00H :"
1229 005604 015 012 .BYTE CR,LF
1230 005606 000 .BYTE 0
1231 005607 .DELSMG:
1232 005607 010 040 010 .BYTE BS,SPC,BS,NUL
1233 005607 .EVEN
;===================================================================
 INBUF: .BLKB  FRAMELEN ; Save space for Message from net
 ENDBUF =.
 NOMOREDATA: .BYTE FALSE ; End of Rx buffer chain flag
 NODATA: .BLKB 1 ; Data read flag
 TXBUF: .BYTE 2 ; Transmit buffer counter
 LONG: .BYTE FALSE ; Long frame flag
 CONTTYPE: .BLKB 1 ; IEEE Control Byte
 DESTADDR: .BLKB 1 ; Destination address
 TIMERCOUNT: .BLKW 1 ; Save space for the timer counter
 SCANTCOUNT: .BYTE 0 ; Save space for the scan timer
 IRDATA: .BYTE 0 ; Store for Interrupt flags
 READFIRST: .BLKW 1 ; Save space for first buffer pointer
 READLAST: .BLKW 1 ; Save space for next buffer pointer
 TOPADDR: .BLKW 1 ; Top of transmit buffers
 RBUF: .WORD 255. ; MAXNO - the maximum number of characters to entered
 RBUFD: .BLKW 1 ; NOENT - the number of characters entered
Stack Area

.EVEN
.BLKW 100.

STACK:

Input buffer area

.RBUFI: .BLKW 1 ; The rest of the memory is for inputs

<table>
<thead>
<tr>
<th>SYMBOL</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACKRES</td>
<td>000020</td>
</tr>
<tr>
<td>ADL10</td>
<td>002416R</td>
</tr>
<tr>
<td>ADL15</td>
<td>002460R</td>
</tr>
<tr>
<td>ADL20</td>
<td>002436R</td>
</tr>
<tr>
<td>AHOLT</td>
<td>000015</td>
</tr>
<tr>
<td>BASE</td>
<td>177140</td>
</tr>
<tr>
<td>BS</td>
<td>000010</td>
</tr>
<tr>
<td>BSIZE</td>
<td>000004</td>
</tr>
<tr>
<td>CBLK</td>
<td>000000</td>
</tr>
<tr>
<td>CPBH</td>
<td>000012</td>
</tr>
<tr>
<td>CBRPL</td>
<td>000013</td>
</tr>
<tr>
<td>CIDEL</td>
<td>001222R</td>
</tr>
<tr>
<td>CIEEXIT</td>
<td>001242R</td>
</tr>
<tr>
<td>CIL05</td>
<td>001134R</td>
</tr>
<tr>
<td>CIL10</td>
<td>001123R</td>
</tr>
<tr>
<td>CIL20</td>
<td>001172R</td>
</tr>
<tr>
<td>CIN</td>
<td>001120R</td>
</tr>
<tr>
<td>CNTRL2</td>
<td>000032</td>
</tr>
<tr>
<td>CONTTY</td>
<td>006320R</td>
</tr>
<tr>
<td>CR</td>
<td>000015</td>
</tr>
<tr>
<td>CR0</td>
<td>000000</td>
</tr>
<tr>
<td>CRL</td>
<td>000001</td>
</tr>
<tr>
<td>CTR0</td>
<td>000006</td>
</tr>
<tr>
<td>CYSKIP</td>
<td>000000</td>
</tr>
<tr>
<td>DACEXI</td>
<td>001450R</td>
</tr>
<tr>
<td>DEL</td>
<td>001077</td>
</tr>
<tr>
<td>DELAY</td>
<td>001262R</td>
</tr>
<tr>
<td>DELMSG</td>
<td>005607R</td>
</tr>
<tr>
<td>DESTAD</td>
<td>006321R</td>
</tr>
<tr>
<td>DL10</td>
<td>001237R</td>
</tr>
<tr>
<td>DM10</td>
<td>002312R</td>
</tr>
<tr>
<td>DOASC1</td>
<td>001436R</td>
</tr>
<tr>
<td>DUMMY1</td>
<td>002262R</td>
</tr>
</tbody>
</table>

**Table:**

<table>
<thead>
<tr>
<th>SYMBOL</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACKRES</td>
<td>000020</td>
</tr>
<tr>
<td>ADL10</td>
<td>002416R</td>
</tr>
<tr>
<td>ADL15</td>
<td>002460R</td>
</tr>
<tr>
<td>ADL20</td>
<td>002436R</td>
</tr>
<tr>
<td>AHOLT</td>
<td>000015</td>
</tr>
<tr>
<td>BASE</td>
<td>177140</td>
</tr>
<tr>
<td>BS</td>
<td>000010</td>
</tr>
<tr>
<td>BSIZE</td>
<td>000004</td>
</tr>
<tr>
<td>CBLK</td>
<td>000000</td>
</tr>
<tr>
<td>CPBH</td>
<td>000012</td>
</tr>
<tr>
<td>CBRPL</td>
<td>000013</td>
</tr>
<tr>
<td>CIDEL</td>
<td>001222R</td>
</tr>
<tr>
<td>CIEEXIT</td>
<td>001242R</td>
</tr>
<tr>
<td>CIL05</td>
<td>001134R</td>
</tr>
<tr>
<td>CIL10</td>
<td>001123R</td>
</tr>
<tr>
<td>CIL20</td>
<td>001172R</td>
</tr>
<tr>
<td>CIN</td>
<td>001120R</td>
</tr>
<tr>
<td>CNTRL2</td>
<td>000032</td>
</tr>
<tr>
<td>CONTTY</td>
<td>006320R</td>
</tr>
<tr>
<td>CR</td>
<td>000015</td>
</tr>
<tr>
<td>CR0</td>
<td>000000</td>
</tr>
<tr>
<td>CRL</td>
<td>000001</td>
</tr>
<tr>
<td>CTR0</td>
<td>000006</td>
</tr>
<tr>
<td>CYSKIP</td>
<td>000000</td>
</tr>
<tr>
<td>DACEXI</td>
<td>001450R</td>
</tr>
<tr>
<td>DEL</td>
<td>001077</td>
</tr>
<tr>
<td>DELAY</td>
<td>001262R</td>
</tr>
<tr>
<td>DELMSG</td>
<td>005607R</td>
</tr>
<tr>
<td>DESTAD</td>
<td>006321R</td>
</tr>
<tr>
<td>DL10</td>
<td>001237R</td>
</tr>
<tr>
<td>DM10</td>
<td>002312R</td>
</tr>
<tr>
<td>DOASC1</td>
<td>001436R</td>
</tr>
<tr>
<td>DUMMY1</td>
<td>002262R</td>
</tr>
</tbody>
</table>
ENDBUF= 006314R
ENQ  = 000005
EOT  = 000032

ITERR = 000200
ITOK  = 000004
ITRAN = 000020

NDL05  002502R
NDL10  002546R
NETDEA 002472R

RSCCHA= 000001
RSCMSG 000716R
RMEXIT 001022R

WIXFCB= 000200
XBUI = 177566
XCSR1 = 177564

. ABS. 000204  000
         006652  001

ERRORS DETECTED: 0

VIRTUAL MEMORY USED: 8247 WORDS (33 PAGES)
DYNAMIC MEMORY: 9820 WORDS (37 PAGES)
ELAPSED TIME: 00:00:48
APPENDIX I

Program Users' Guides

I.1 Using EMS

To use this program proceed as follows. Make a copy of the EPROMs onto the EMS disk. (The EMS disk presently contains a copy of the EPROMs used in the micro-computers.) The files must be named EPDIR.COM for the first one and then EPl.COM, EP2.COM and so on. To run the program type:

EMS<CR>

This will give the main menu. If M is chosen, a display of the program map is sent to the console. If D is selected then the user is prompted for the name of the file to delete. The EMS program will either delete the program specified or, if there is no such program in the directory, it will inform the user. If the program is deleted, the user will be instructed to re-program the EPDIR EPROM with the contents of EPDIR.COM which is now the updated version of the directory. If A is chosen, then the user will again be prompted for the name of the file. Should the specified program already be present on the pseudo-disk, then the user is given the choice of replacing the old program or aborting the addition utility. The EMS program will then load the program from the specified file to the EPn.COM file, giving the user a list of the EPROMs that need reprogramming. Typing a R will return the user to CP/M.
I.2 Using the TAC User Routines

The user routines may be used by any 8085 program run on the SABus computer. The program must first be loaded from the EPROM card by typing:

*LAN<CR>

to which there are two possible responses. The first is the duplicated address message:

Duplicate TAC addresses

In this case the EPROM's have been programmed with the incorrect TAC address value. This address is found at address 0025H and should be different for each TAC. The second possible response is:

TAC initiated

This may take a while to appear. If no other TAC is on the network, it will wait until another one is initialised. This message means, therefore, that the TAC has been initialised and that it is currently taking part in a logical ring.

All the entry points are entered by jumping to the appropriate offsets with the registers set to the required values. To enable direct control of the WD2840, the registers may be accessed directly by the 8085 IN and OUT commands. The TAC is situated at address 70H in the I/O page of the 8085 but is mini-jump selectable to any address on a 16 byte boundary.
I.3 Using the CP/NET Programs

The TAC User routines must first be loaded as explained above and then the CP/NET programs may be loaded from the EPROM card when CP/NET is run from the SABus computers by typing:

*CPNET<CR>

The rest of CP/NET is as specified in the CP/NET Users guide.

I.4 Using the PDPLAN Program

To use this program it is necessary to load the program down from the PDP to the FALCON computer. This is done in two stages. The first section is to load down, under ODT control, a loader program which will enable the fast transmission of the user program. This loader program will read the start address of the block and the length of the block from the first two words received and then write the subsequent data to the correct place in memory.

There are two programs which run on the PDP 11/23 and which are used to send the programs. The first, PDPLD, will send the data in a way that ODT would expect the user to type it in. The second will start the loader program previously sent and then write the user program into memory.

To run these programs, set the line connected to the FALCON to NOECHO and SLAVE mode, and then type:
RUN PDPLD<CR>

on another terminal logged on under 100,7. This program will prompt the user for the file name and the terminal of the FALCON. The program name should be LD.OBJ. Once this is complete, the user should then type:

RUN DNL<CR>

to run the download program. This too will prompt for a file name and the destination terminal. The file would be PDPLAN.OBJ. Once this is complete, the user may run the PDPLAN program by typing

500G

on FALCON. The response will be one of two messages. Firstly, if the network is active then:

WELCOME TO PDPLAN

TESTING FOR DUPLICATE ADDRESS

will be displayed. If there is a duplicate address then the following message will appear:

ADDRESS DUPLICATION - CHANGE MY ADDRESS

and the program will stop. If no address duplication occurs then the following messages appear:
NO ADDRESS DUPLICATION - ENTERING LOOP

READY FOR DATA EXCHANGE

TYPE: T TO TRANSMIT A FRAME
R TO RECEIVE A FRAME

The TAC is now in the logical loop and ready for data exchange. The user may now transmit a frame by choosing T, in which case the user will be prompted for a destination address by the following message:

TO WHICH TERMINAL MUST THE MESSAGE BE SENT?

Ø1H TO FEH - TO SEND TO A SPECIFIC TERMINAL
FFH - TO BROADCAST
ENTER THE HEX VALUE >

Once the hex value has been entered, which may be corrected by the <DEL> key, the user will be informed that it was incorrect if that is the case, and will be prompted for a new value or, if the address was valid, then the user is prompted for the message with the following message:

NOW ENTER THE MESSAGE TO BE SENT (Z TO TERMINATE):

The Falcon will then accept up to 650 characters which will be sent. Once the console displays the transmit or receive message again it means that the frame has been sent.

To receive a message, the user may type R to the transmit or receive message, but each time the TAC receives a message it will automatically be displayed.
Appendix J

A detailed discussion on the choice of the U.C.T. LAN architecture.

(This is a copy of Chapter 2 of a previous draft.)
CHAPTER 2

The U.C.T. LAN Architecture

No one LAN architecture is the best for all applications. It is thus necessary to carefully examine each network's requirements and design a LAN specifically for that application.

This chapter attempts to show that the Token-passing bus LAN was the best architecture for the U.C.T. LAN.

2.1 The U.C.T. LAN Requirements

The requirements for this LAN are as follows:

1. Low cost.
2. Minimal cabling.
3. A maximum of 30 stations.
4. 1 Mbps transmission rate.
5. An overall physical path of not more than 800 metres.
Any one of a large number of architectures would fulfil these requirements, e.g. slotted-ring, CSMA/CD bus, token-passing rings or busses and a variety of star networks. In order to provide the most cost-effective and long-term design, Wittlin and Ratner [27] state that one should choose an architecture that adheres to some standard. This ensures that for future additions and modifications there are a number of possible equipment suppliers, thereby guaranteeing the availability of new equipment at (hopefully) reasonable prices.

The choice of the architecture was thus limited to one of those specified by the IEEE 802 standards which are generally considered to be the emerging LAN standards.

2.2 Overview of the IEEE 802 Architectures

Up to the beginning of this decade computer networks were most often designed around the International Standards Organisation's Open Systems Interconnection Reference Model (ISO/DIS 7498), known as the ISO RM, which defines computer intercommunication. With the increased demand for computer communication caused by the development of LSI technology and the subsequent creation of the micro-processor, manufacturers began designing local computer communications networks based on their interpretation of the ISO RM. This
The U.C.T. LAN Architecture  
- Overview of the IEEE 802 Architectures

led to a proliferation of LAN architectures. The IEEE appointed the 802 committee with the task of attempting to standardize on certain LAN architectures. The results of their work is a suite of standards collectively called the IEEE 802 standards.

The general aim of the 802 standards was to transform the ISO RM into a model that applies directly to local area networks. This was achieved by re-defining the lowest two ISO RM layers, the Physical and Data-Link layers as the Physical (PHY), Media Access Control (MAC) and Logical Link Control (LLC) layers. The PHY layer controls the physical signalling over the network, the MAC layer controls the access to the shared medium, and the LLC layer is in charge of the logical transmission of data over the link.

![Diagram of ISO RM and IEEE 802 models]

Figure 2.1 - The ISO Model and the IEEE 802 Model

Very early in its existence the IEEE 802 committee realised that it was unable to standardize on a single LAN architecture. This led to the formation of a number of sub-committees to examine the different networks. By the
beginning of 1985 standards defining four architectures had been drafted, namely:

1. ANSI/IEEE 802.3 - 1985 defining a baseband CSMA/CD bus network
2. ANSI/IEEE 802.4 - 1985 defining a broadband and
3. baseband Token-passing bus network
4. ANSI/IEEE 802.5 - 1985 defining a baseband Token-passing ring network

2.2.1 IEEE 802.3 - CSMA/CD Bus Network

Carrier Sense Multiple Access with Collision Detection (CSMA/CD) is a technique which enables a number of nodes to transmit information (at random intervals) over a common medium. Random access is achieved, firstly, by sensing the network for activity (carrier sensing) before transmitting and, secondly, once the network is found to be unused and the station has begun its transmission, by providing a "listen-while-talk" facility (collision detection) which compares the data being sent to that on the network. Inequality between these signals indicates a frame collision, caused by more than one station transmitting simultaneously.

To allow time for the receiver to process the frame and
to wait for the dissipation of any transients on the network, the IEEE 802.3 standard defines a period of quiet between frame transmissions called the "inter-frame" delay. To enforce this period, the station must continuously sense the network and begin a timer on detection of no transmission, i.e. the end of a frame. This prevents the transmitter from attempting to access to the network before the end of the inter-frame period.

When data is presented to the MAC layer for transmission, it constructs a frame by adding the header fields, checksum and flags, and places the frame into a transmitter queue. Transmission of these frames is postponed if another station is using the network or if the inter-frame period has not elapsed since the previous transmission. Once this period has expired, transmission is initiated. Because no station will begin its transmission whilst the network is in use, a collision must occur within the end-to-end propagation time for the network and thus be detected at the transmitting stations before the end of two such propagation times. Collision detection is also the reason for specifying a minimum frame length to be longer than twice the end-to-end propagation time for the network.

When a collision does occur, the transmitting station "jams" the network by issuing a number of jamming bits (48
in the IEEE 802.3 case) thus ensuring that all participating stations are aware of the collision. Jamming is required because the analogue collision detection circuitry may not detect a very short collision, e.g. if a second station began its transmission momentarily before receiving the first station's frame and instantly stopped on detection of the collision.

![CSMA/CD Bus Operation Diagram](attachment:image)

Figure 2.2 - CSMA/CD Bus Operation

After the detection of a collision, contention between the responsible stations is resolved by delaying their re-transmission attempts by a random number \( r \) of "slot times". This slot time is the maximum time required for all the stations on the network to detect a collision and must be greater than twice the network's end-to-end propagation time plus the network jamming time. It is defined as $512$
The random period \((r)\) is calculated by a controlled randomisation process called "truncated binary exponential backoff", and is an integer which lies between 0 and \(2^k\), where \(k\) is the smaller of the retransmission count and 10. A station will attempt to re-transmit the frame a maximum of 16 times after which it will inform the station management and terminate. Only once a frame has been successfully transmitted, will a new frame be sent.

The IEEE 802.3 standard defines the following parameter values:

1. Slot time = 512 bit times
2. Inter-frame gap = 9.6 usec
3. Jamming bits = 48
4. Minimum frame size = 64 octets (512 bits)
5. Maximum frame size = 1 518 octets
6. Maximum end-to-end distance = 2.5 km
7. Data Rate = 10 Mbps.

2.2.2 IEEE 802.4 - Token-bus Network

Token-passing is a distributed polling access method used to provide controlled sharing of a common medium by a
number of terminal devices.

Essentially the Token-passing operation is this: the right to the network is passed from station to station around a logical ring on the bus network in the form of a unique control frame called the "token". The station that is addressed by the token may transmit up to a pre-defined maximum number of frames or simply pass the token to its successor. Token rotation is in descending order of station addresses, except that the lowest station passes the token to the station with the highest active address in order to complete the logical ring. Each station keeps a record of its own address, its successor's address and its predecessor's address. By fixing the rotation of the token (to either descending order, as in the IEEE case, or ascending order which the WD2840 uses), the quick inclusion of new stations and the re-establishment of the logical ring after failure, is facilitated.

![Diagram of the Logical Ring on a Token-bus Network](image-url)

**Figure 2.3 - The Logical Ring on a Token-bus Network**
To prevent a single station transmitting for a long period of time ("network hogging"), each station keeps a record of the time that it has held the token. By comparing this value to a fixed maximum token hold time, a station may determine if there is enough time to transmit a queued frame or if it must pass the token.

The IEEE 802.4 standard also defines an optional priority structure which consists of 8 priority levels or service classes. The top four, service classes 4 to 7, are for "synchronous data" which require a guaranteed transmission time, while frames of the lower 4 service classes, 0 to 3, use the residual bandwidth. The service classes are paired to form 4 access classes, where service classes 0 and 1 form access class 0, 2 and 3 form access class 2, 4 and 5 form access class 4 and service classes 6 and 7 form access class 6. Each access class is given a "target" token rotation time; if the token returns after passing around the logical ring and the target time has not been exceeded, a frame from the particular service class's queue may be sent. Thus packets may be sent at the same access class, i.e. have the same target rotation time, but the LLC layer may still define which packets must be sent first, by placing them in the higher service class's queue.

The highest access class, class 6, uses the maximum
token hold timer to determine if it is permitted to send a frame. Should all the frames from service class 7 and 6 be sent and should time still be available on the maximum token hold timer, the token will be passed within the station, to the next service class. The new service class will determine if the token rotation time is less than its target value and, if so, transmit one or more queued frames, but after each frame ensure that there is time available on the target rotation timer. The token will pass to the next service class only once transmission of all queued frames is complete or if the target token rotation time has been exceeded. This continues down to the lowest service class after which the token will be passed to the next station. (A station not using the priority structure, will transmit frames with an access class of 6, and so use only the maximum hold timer.)

Figure 2.4 below illustrates this structure by way of an example. The station under consideration has three service classes, 7, 3 and 1, which fall into access classes 6, 2 and 0 respectively. The maximum hold time is set to, say, 1000 octet times. The target rotation time (TRT) for access class 2 is 2000 and access class 0 is 1500. The station has two frames of 200 octets queued for service class 7, one frame of 500 octets for service class 3 and one frame of 128 octets for service class 1. When the token is
received after passing around the logical ring, it has taken 1000 octet times. The two highest priority frames are sent, leaving 600 octet times on the maximum token hold timer for the lower levels. The token is thus passed to the service class 3 sub-station, which calculates that there are 600 octet times available for transmission based on its TRT (i.e. \(2000 - (1000 + 400)\)), and so it transmits its frame. Only 900 (i.e. \(400 + 500\)) octet times have been used since the token was received, which is within the maximum token hold time, thus the token is passed to level 1. This level discovers that its target rotation time has been exceeded (i.e. \(1500 < (1000 + 400 + 500)\)). The token is therefore passed to the next station without the transmission of any level 1 frames.

![Figure 2.4 - Priority Transmission on the Token-bus](image-url)

This priority frame handling structure makes the IEEE Token-bus popular for voice and control applications where
large delays cannot be tolerated. Such frames can make use of the "synchronous" data transmission.

In a real network there are a number of conditions which require special procedures for their handling. These conditions are: new station inclusion, initialization of the logical ring, action on discovery of a lost token and station removal.

(a) New Station Insertion:

A new station may only be included in the logical ring when invited to do so by one of the already active stations. This is achieved by having the active stations issue "Solicit Successor" frames at regular intervals. A "Solicit Successor" frame specifies a range of station addresses that may respond (with a "Set Successor" frame) in order to be included in the logical ring. During the successor soliciting phase, the token-holding station attempts to find a new station whose address falls between its own and its successor's addresses. It thus looks for a station with the highest address in the specified range, or, if the soliciting station is the lowest active station, it looks for a new station with lower address or one that has a higher address than the highest active station.
It is possible that more than one station in the address range, specified by the solicit successor frame, will require inclusion in the logical ring. To ensure that only the station with the highest address in the range is included, a contention resolution process, which makes use of "response windows", is defined. A response window is a fixed period, calculated from the completion of a frame to the start of the responding station's frame. Its purpose is to define the starting time of a response frame and thereby enable the sort process defined below.

If the token-holding station receives a garbled message, it will assume that more than one station has responded with a "Set Successor" frame and will thus issue a "Resolve Contention" frame. Those stations which responded to the "Solicit Successor" frame will transmit a second "Set Successor" frame, but this time delayed by 0, 1, 2 or 3 response window periods after the end of the token holder's "Resolve Contention" frame. This delay is computed by complementing the two most significant bits in the station's address, i.e. if the two bits are 00, 01, 10 or 11 the transmission is delayed by 3, 2, 1 or no response windows respectively. A station will only respond if no other station had already begun its transmission.
If the token-holding station is still unable to interpret the frame, it will issue a second "Resolve Contention" frame. Only those stations that sent frames in response to the previous "Resolve Contention" frame will respond again, but this time their transmission will be delayed by a value calculated, in the same way, from the second most significant pair of bits in the address. This procedure will continue until a valid "Set Successor" frame (which must be issued by the station with the highest address in the specified range) is received by the token holding station.

The stations excluded by the sort process will be
(b) Initialization of the Logical Ring:

The initialization process consists of two phases, the first is the selection of a single initializer (as more than one station could attempt to assume this role). The second phase establishes the logical ring on the network using a sort-by-address process described in (a) above.

On network start-up, or after the network has been inactive for 6 or more "slot times", an attempt will be made to (re-)establish the logical loop. The maximum time required for a MAC layer to receive an immediate MAC layer response, i.e. one generated by the remote MAC layer itself and not by its higher levels, is called the "Slot time" ($T_s$), which is equal to one "Response Window", and is defined by IEEE as follows:

$$T_s = \text{INT} \left[ \frac{2(T_p+T_{sta})+T_{sm}}{T_{sym}} + \frac{7}{8} \right] \text{[Octet times]}$$

Where $T_p$ is the end-to-end propagation time for the network.
The U.C.T. LAN Architecture
- Overview of the IEEE 802 Architectures

\[ T_{\text{sta}} \] is the time between the data being ready and the start of transmission.

\[ T_{\text{sm}} \] is the safety margin (at least 1 symbol time long).

\[ T_{\text{sym}} \] is the symbol time. (1 µs for a 1 MHz network)

On start-up each station has its "Inactivity Timer" set to 7 slot times. Once the logical ring has been established, the node with the lowest address resets its value to 6 slot times.

The expiry of the "Inactivity Timer" prompts the station to claim the token by transmitting a "Claim Token" frame. Assuming that more than one station transmitted such a frame, which could occur at network start-up or if the lowest station failed, the IEEE 802.4 standard provides a means of isolating a single station, namely the highest station that sent a "Claim Token" frame.

This contention resolution process, which is a sort based on the length of the "Claim Token" frame, is achieved by each potential initializer setting the length of its "Claim Token" frame's data field to reflect the value of a pair of bits in the address. The first "Claim Token" frame's data field will be 0, 2, 4 or 6 slot times long
depending on the value of most significant pair of bits in the address, i.e. if the values of the top two bits were 00, 01, 10 and 11, the data field would be 0, 2, 4 and 6 slot times long respectively. Once the frame has been sent, the station will wait one slot time (for the propagation of its own frame and other station's frames, sent at the same time) and then sense the network for activity. Only those initializers that sense no activity are allowed to continue vying for initializer status; the other stations must remain silent.

The contention resolution process continues by sending a second "Claim Token" frame. This time the length is related, in a similar way, to the second most significant pair of bits in the address. This procedure continues until all the bits in the address have been used, as well as two random bits. (The random bits allow for the error condition where two stations have the same address.) The station which sends the last "Claim Token" frame and hears nothing has the token and assumes the role of initializer.
The second phase, that of establishing the logical ring, is achieved by the sort-by-address process described in (a) above.

(c) Station Removal or Failure:

A station passing the token is responsible for ensuring that network activity follows the transmission of the token. No activity prompts the re-transmission of the token. If there is still no response after the second token transmission the token-holding station must attempt to find the next lower station by issuing a "Who Follows" frame.
which is an attempt to make contact with the failed station's successor. This frame requests the station whose predecessor address is equal to the value in the data field to respond with a "Set Successor" frame. If such a station is active on the network, which will be the case unless both of the two subsequent stations have failed, it will respond with a "Set Successor" frame and the ring will be re-established.

No response to a second "Who Follows" frame will lead to the issuing of a "Solicit Successor" frame in order to find any other active stations. If a response is received from one or more stations, the logical ring is formed as described in (a) above. A silent network after two attempts means that either the station has become deaf or the network is unusable, and so the token holding station will enter the listen-only state.

If a station fails while holding the token, the lost token recovery procedure (described in (d) below) will be invoked.

To permit orderly exclusion of a station from the logical ring, a station may send a "Set Successor" frame, containing its successor's address, to its predecessor. On the next token rotation the station will be excluded.
(d) Token Loss and Duplicate Token:

If the Token is lost through a station failure, recovery will be initiated by the expiry of an inactivity timer. The recovery procedure follows the strategy described in (b) above.

If a station which has the token detects a transmission that is not a response frame, it will assume that there is a duplicate token on the network. Under this condition it will not pass the token and will wait for a new one before transmitting any further frames.

2.2.3 IEEE 802.5 - Token Access Ring

In the Token-ring network, each station is connected to two others by means of a point-to-point link to form a closed ring. This connection is made by means of a Network Interface Unit (NIU) which ensures that the network is not halted should a station become inactive.
Frames are transmitted around the ring by being relayed from one node to the next. To ensure ordered access to the common medium, a Token frame is also circulated. A station, which has data to transmit, gains access to the network by setting the busy/free bit in a passing token, thereby converting the token into a start-of-frame marker to which the frame is appended. Because the IEEE 802.5 Token-ring is a single token protocol, i.e. only one token may be present on the network at any time, it is the responsibility of the token holding station to issue a new token only once the frame header has circumnavigated the network, providing that the station has no more queued data or it has sent the maximum number of frames per token reception.

A priority facility, incorporating 7 priority levels (1 - 7), is provided on the Token Ring. When a station receives a high-priority frame from its LLC layer it will
transmit the frame on receipt of the next free token of lower or equal priority. If the network is busy, the station will set the appropriate "reservation" bits in the header of a passing frame, indicating the level of priority required. On receipt of this header, the token holding station will issue a high-priority token, allowing access to frames of that or a higher priority only. In order to prevent a high-priority token from circulating continuously, the station that first issued the high-priority token is responsible for returning it to the priority it was received at, after a single rotation.

Frame reception is achieved by the NIU copying all the passing frames into an internal buffer. The destination address is then compared with its own address and if it corresponds the frame is passed to the computer via the LLC layer.

Ring maintenance is achieved by assigning to one of the active stations the task of "Monitor". On start-up each station enters stand-by monitor mode. After a period of network inactivity, defined by the "Inactivity Timer", any station may issue a "Claim Token" frame in an attempt to become the active monitor. To ensure that only one monitor exists, a station vying for monitor status defers to a station with a higher address. The receipt of a "Claim
Token" frame with its own address as the source address means that a station has been successful in its attempt to become the Monitor. (A "Claim Token" frame with a lower source address than the station's own address is ignored.) If an active monitor receives a "Claim Token" frame, an error has occurred. To facilitate quick recovery, it will cease to be the monitor and return to stand-by status.

Figure 2.8 - Simplified Token-ring Monitor State Diagram

The Monitor station has the following tasks:
1. It must issue a new token when the network has been inactive for a specified period.
2. It must check for and recover from continuously rotating high-priority tokens.
3. It must prevent the continuous rotation of a frame.
2.2.4 The WD2840 Token-bus

The Token-bus has often not been considered as a viable network architecture because of the complexity of its MAC layer. Recently, though, Western Digital Incorporated introduced the WD2840 Token-access controller which is a single chip that implements the Token-bus MAC layer. The WD2840 was used in this project, but unfortunately its implementation of the MAC layer differs from the IEEE 802.4 specification in several respects, namely:

1. Direction of token rotation.
2. Frame format.
3. Dynamic operation.

1. Direction of Token Rotation:

The direction of the token rotation is opposite to that of the IEEE 802.4 standard, in that the token is passed from a station with a lower address to one of with a higher address. The last station, with the highest address, will then pass the token back to the first station, i.e. the one with the lowest address, and so complete the logical ring.
2. Frame Format:

Firstly, the fields for both the destination and source addresses are only 1 octet long, and not 2 or 6 as defined by IEEE.

Secondly, the frame check sequence is a 16-bit field rather than a 32-bit field.

Thirdly, all frames contain an access control field. This field performs two functions: either it requests an immediate acknowledgement to the frame being sent (when it has a value of 255) or it "piggy-backs" the token by containing a valid address, i.e. between 1 and 254. (A value of 0 has no meaning or effect.) A data frame can therefore simultaneously transfer data and pass the token to the same or different stations.

The shorter address field and FCS reduce frame overhead, thereby improving the potential throughput. These fields limit the WD2840 to non-IEEE 802.4 networks and, because of the shorter FCS, its reliability is lower.
3. The Dynamic Operation:

(a) Initialisation of the Logical Ring:

When the inactivity timer (or the Network Dead timer) expires and the station has been assigned the task of initializer, the station will assume that the token has been lost or that the network has just been powered up and will attempt to (re-)establish the logical ring.

On network start-up, the first station that is powered up will attempt to establish the logical ring. This is achieved by a "scan operation" where each station sets its successor's address value to its own address plus one. An attempt is then made to pass the token. If no response is forthcoming, the station will increase the value and attempt transmission again. Once a response is evoked, the station will save the address as that of its new successor. As each station receives the token it will scan for its successor until the initialising station again receives the token.
(b) Token Loss:

Recovery from a lost token is achieved as follows: When the token holding station passes the token it listens for a response. If nothing is received in the required response period, it re-transmits the token. If there is still no response it will scan for a new station, using the procedure in (a) above.

If a station fails while holding the token, after a period of time defined by the shortest Network Dead timer, a station will set its successor address to its own address plus one and begin the scanning procedure as described in (a) above.

(c) Station Insertion:

The WD2840 controller has no means of calculating when to solicit for a new successor. It is thus the responsibility of the host computer or the interface hardware to initiate the scan cycle. This is achieved by setting the station's successor address to its own address plus one. On the next token pass it will search for a
successor. A new station, having an address which falls between the station's own address and its successor's address, will thus be included in the logical ring.

(d) Station Removal or Failure:

The WD2840 does not define a "Set Successor" frame as it keeps no record of its predecessor's address. A station will thus be removed after ignoring the next token, resulting in a scan for the new successor by its predecessor.

If the station fails while holding the token, the expiry of a Network Dead timer will invoke the logical ring re-establishment procedure described in (a) above.

2.3 A Comparison of the Architectures

In this section the IEEE architectures and a Token-bus network using the WD2840 TAC are discussed and compared in terms of performance, reliability and physical considerations.
2.3.1 Performance

The performance of a LAN is measured in terms of throughput and queueing delay. The throughput is the rate at which user data is transported across the LAN, while the queueing delay is the time between the data being presented to the first station's NIU and its presentation to the destination node.

(a) Throughput:

The IEEE 802.3 CSMA/CD Bus Throughput:

The distributed and random nature of CSMA/CD means that there is a finite probability that more than one station will attempt to access the common medium at any particular time. The maximum throughput of this network is calculated assuming that all the active stations will always have data to send, i.e. a new frame is immediately queued once a frame has been sent. After the successful transmission of a frame, all the active stations will wait the required inter-frame period and then begin transmitting. Should more than one station be active, a collision will occur. Frame re-transmissions are then controlled by the truncated binary exponential backoff method described in the previous section.
To formulate the throughput efficiency, it is assumed that the probability of any one station transmitting immediately after a collision or successful transmission is \( P \) and the probability of that station not transmitting in the first slot is \( 1 - P \). The probability of all \( Q \) active stations transmitting immediately after the collision is therefore \( P^Q \), and the probability of all stations deferring transmission is \( P^Q \). The probability of all but one station deferring transmission is:

\[
x = QP(1-P)^{Q-1}
\]  

The truncated binary exponential backoff random delay generator attempts to give \( P \) a value of \( Q^{-1} \), which in turn maximizes \( x \). The probability of only one out of \( Q \) stations transmitting in one slot time is \( A \), and may be defined as:

\[
A = x_{\text{max}} = \left(1 - \frac{1}{Q}\right)^{Q-1}
\]

The average number of slots wasted before a successful transmission occurs may now be calculated. The probability of any of the \( Q \) active stations transmitting immediately after the collision is simply \( A \). The probability of a successful transmission after a single slot is \( A(1-A) \). The probability of a successful transmission after \( i \) slots is
thus: \( A \|1 - A\|^i \). The mean of the sum of all the probabilities, i.e. the sum \( A\|1 - A\|^i \) for values of \( i \) from 1 to infinity, will be the mean delay between successful transmissions.

This mean, \( W \), is thus the average waiting time (in slot times) and is:

\[
W = \sum_{i=1}^{\infty} A\|1 - A\|^i
\]

\[
= \frac{1 - A}{A}
\]

... (3)

To ensure that the data presented to the MAC layer arrives at its destination correctly, it is placed into a frame. The creation of this frame requires the use of additional bits which in turn reduce the effective throughput.

Two other factors combine to further reduce the efficiency of the network, these being the inter-frame delay and the minimum frame length required by CSMA/CD for collision detection. The inter-frame delay reduces the efficiency because no data transfer can take place during its duration. The minimum frame length does so because, if less data is supplied than is required to fill the minimum
frame, the difference is made up of padding characters.

The throughput efficiency of the CSMA/CD network is thus:

\[ S = \frac{T_d}{|T_d + W*T_s + T_p + T_f + T_i|} \]  

... (4)

Where \( T_d \) is the average time taken to send the LLC data.
\( T_s \) is the slot time.
\( T_p \) is the time spent sending padding characters.
\( T_f \) is the time spent sending framing characters.
\( T_i \) is the inter-frame delay.
\( W \) is the waiting time in slot times defined by (3).

The inter-frame gap and the framing bits are defined for the IEEE 802.3 network. The variables are the packet size and network propagation time. Using equation (4) and the following parameter values, the maximum throughput for the IEEE 802.3 network may be calculated:

1. Transmission speed = 1 Mbps.
2. Number of active stations = 30.
3. Transmission path = 800 m.
4. Frame length = 256 octets.
5. Frame overhead = 138 bits.
6. Inter-frame delay = 9.6 usec.

The above values allow for a transmission rate of 452.37 frames per second which corresponds to a maximum throughput efficiency of 92.65%. All the other parameter values, with the exception of the frame length, are fixed for the U.C.T. network. Figure 2.9 below illustrates the effect of the value of this parameter on the performance of the CSMA/CD network.

**The IEEE 802.4 Token-bus Throughput:**

The predominant cause of inefficiency in the Token-bus architecture is due to the rotation of the token around the logical ring. This is clear when one considers that a new token has to be generated and transmitted across the network for each token pass. If the token is $B$ bits long, it will take $B/C$ seconds to be transmitted from a station onto a network with a capacity $C$ bps, and take $B/C$ plus the propagation time ($T_p$) to reach the most distant station. The Token rotation time ($T_r$), when only the token is circulating and no data is being transmitted, is thus the token transmission time per station multiplied by the number
of active stations on the logical ring \( (Q) \).

\[
T_r = Q(B/C + T_p)
\]  \hspace{1cm} \text{(5)}

If only one station is transmitting while the rest are simply passing the token, there will be a lower throughput because of the token regeneration at each station. It is clear that if more stations are transmitting information, or if more data is allowed per token hold, that the throughput will be proportionally higher.

As in the CSMA/CD case, the frame overhead will reduce the information throughput of the network. The efficiency, \( S \), of the token-bus network with transmitting nodes is thus:

\[
S = \frac{T_d}{T_d + \left| T_r/Q + T_f \right|}
\]  \hspace{1cm} \text{(6)}

Where \( T_d \) is the average time taken to send the data.
\( T_r \) is the token rotation time defined by (5)
\( Q \) is the number of nodes.
\( T_f \) is the time spent sending framing bits.
The values of $T_d$ and $T_r$ are adjusted where multiple frames may be transmitted per token hold.

The two major causes of inefficiency are the number of nodes active on the logical ring that are not transmitting data frames, and the end-to-end propagation time. As the number of frame level overhead bits are fixed by IEEE, the variables are the number of non-transmitting stations active on the logical ring, the length of the network which effects the propagation time, the speed of transmission and the packet size.

Equation (6) was used to calculate the maximum throughput for this network, using the following parameter values:

1. Transmission speed = 1 Mbps.
2. Number of active stations = 30.
3. Transmission path = 800 m.
4. Frame length = 256 octets.
5. Frame overhead = 108 bits.
6. Token frame = 9 octets.

These values allow a maximum of 441.18 frames per second thus giving a maximum throughput efficiency of 90.35%.

A smaller frame length means that a greater percentage
of the time is being used to transmit frame overhead as opposed to user information, thus reducing the throughput efficiency. Mathematically this means that the value of $T_d$ in equation (6) is reduced which, in turn, affects the ratio of control to user data and thus the efficiency. Figure 2.9 below illustrates the effect of frame length on the throughput for the Token-bus network.

The IEEE 802.5 Token-ring Throughput:

In this architecture it is, again, the rotation of the token that is the major cause of inefficiency. However, in this case the network propagation time includes the NIU latency, i.e. the delay imposed by the NIU between receiving and relaying a bit. Because the IEEE 802.5 Token-ring is a single token network, which means that a station must receive its own start-of-frame marker before issuing a new free token, the maximum throughput is dependent on whether the frame transmission time is longer or shorter than the network propagation time.

It is clear that if the frame transmission time is shorter than the propagation time, the station will have to transmit idles until the start-of-frame marker returns, thereby reducing the throughput. In this case the maximum throughput efficiency function is given by:
The U.C.T. LAN Architecture
- A Comparison of the Architectures

\[ S = \frac{T_d}{|T_r + T_f/Q + T_f|} \] ...

(7)

Where \( T_d \) is the average time taken to send the data.

\( T_r \) is the token rotation time defined by (8)

\( Q \) is the number of nodes.

\( T_f \) is the time spent sending framing bits.

\[ T_r = T_p + Q \times T_l \] ...

(8)

Where \( T_p \) is the propagation time for the whole ring (excluding NIU latency).

\( Q \) is the total number stations on the ring.

\( T_l \) is the NIU latency.

For the case where the frame transmission time is longer than the network rotation time, (7) is redefined as follows because the limiting factor is now the transmission time of the frame and not the rotation time:
The maximum throughput may now be calculated from equation (9) using the following parameter values:

1. Transmission speed = 1 Mbps.
2. Number of active stations = 30.
3. Transmission path = 800 m.
4. Frame length = 256 octets.
5. Frame overhead = 104 bits.
6. NIU latency = 1 bit time.

Since the frame is 256 octets long, which equals a transmission time of 2.152 msec (for the data plus framing bits), while the network rotation time is only 31.667 usec (for 29 stations plus 800 m of cable), the maximum
throughput calculations used equation (9) instead of (7). The maximum throughput efficiency is therefore 95.12%.

The major contributors to the inefficiency are the latency of the NIU, the number of nodes that are not transmitting data and the packet size. The packet size determines the number of frames required, and thus the number of framing bits, for the supplied user data.

The figure 2.9 below shows the dependence of the maximum throughput on the packet size of the frame for all the above networks.

![Figure 2.9 - Maximum Throughput versus Packet Size](image)
As was expected, the maximum throughput improves for a larger packet size as the proportion of framing overhead to information bits is lower. Stallings [28] shows this trend for a larger range of packet transmission times.

Figure 2.9 also shows that there is little difference in this case between the maximum throughput for the three networks for a particular packet size and that the greatest effect is at low packet sizes. It is the Token-bus network that displays the lowest throughput due to the extra control information required because the token is regenerated at each node.

(b) Delay

Throughput is related to the ratio of information to the sum of the control and overhead bits on the network. The delay, on the other hand, might be defined as the amount of time taken by all seven network layers to transmit a unit of data between two application tasks. Included in this delay is the time taken to transfer the data from the application task to the physical drivers, the time taken for it to be transmitted across the network, and the time required to pass it to the remote application task. In this section only the delay of the access control section of the NIU is considered. The queueing delay may thus be defined
as the time between data being presented at the LLC/MAC layer interface of the local computer to its presentation to the LLC layer by the MAC layer at the remote computer.

This delay is determined by, firstly, the rate at which the data is supplied to the MAC layer, and secondly, the capacity of the network.

Firstly, with regard to the rate at which the data is supplied, there is a cut-off point, at the maximum throughput rate of the network, after which the network imposes infinite delay on the supplied data. In practice the delay increases slowly at first and then approaches infinity as the maximum throughput is neared. This means that the load induced delay of a unit of data is determined by the proximity of the presented load to the maximum throughput of the network. (The maximum throughput is that value calculated for the different access methods in the previous section.)

The second part of the delay is due to the time taken by the network to carry the data frame from the source to the destination, i.e. the transmission delay. This is simply the length of the frame divided by the capacity of the network plus the propagation time for the network. The longer the frame, the longer the transmission delay as the
two are linearly related.

Bux, in reference [19], gives the transfer delay functions for the Token-ring, Slotted-ring, CSMA/CD and an ordered-access bus using the Multilevel Multiple Access (MLMA) scheme. The results for the CSMA/CD and Token-ring are given below, as well as the transfer delay function for Token-bus network, which was derived from the Token-ring delay function.

The CSMA/CD Transfer Delay Function:

Equation (10) below is based on a derivation by Lam [20]. This equation assumes that the CSMA/CD network does not become unstable and so produce infinite message delays under collision conditions but uses an adaptive algorithm, such as the truncated binary exponential backoff method, to ensure network stability. The arrival times of the MAC layer packets are assumed to be exponentially distributed. The transfer delay \(D\) for the CSMA/CD network may thus be described as:
\[ D = \frac{\lambda [E[T_P^2] + 4e + 2\tau E[T_P] + 5\tau^2 + 4e(2e-1)\tau^2]}{2(1-\lambda)[E[T_P]] + \tau + 2e\tau} + E[T_P] \]

\[ + 2\tau e - \frac{[1-e^{-2\lambda\tau}]\{\frac{2}{\lambda} + \frac{2\tau}{e} - 6\tau\}}{2\{F_p^*(\lambda)e^{-\lambda\tau}/e - 1 + e^{-2\lambda\tau}\}} + \frac{\tau}{2} \]

... \{10\}

Where \( \lambda \) is the packet arrival time.

\( E[T_P] \) is the mean service time for the packet, both header and data.

\( \tau \) is the propagation time for the network.

\( F_p^*(\lambda) \) is the Laplace transform of the probability density function of the packet service time \( T_P \).

This equation was used in figure 2.11 below to illustrate the dependence of delay on load for the CSMA/CD network with the following parameter values:

1. Number of active stations = 30.
2. Data path length = 800 m.
3. Data rate = 1 Mbps.
4. Packet size = fixed length of 256 octets.
The Token-ring Transfer Delay Function:

A Token-ring network may be modelled as shown in figure 2.10 where there are as many queues as there are active stations attached to the network. The network services the queues in a fixed rotation, with a switching delay of \( \tau \) between each queue's servicing. This switching delay is the sum of the NIU latency and the propagation delay to the following station. The packets are queued at a rate of \( \lambda_t \).

![Figure 2.10 - The Token-ring Model](image)

In the relationship derived by Bux [19], based on the discrete time approach model described by Konheim and Meister [26], both the packet arrival times, \( \lambda_t \), and the switching times, \( \tau \), are equal. Equation (11) assumes that the discrete time interval approaches 0, and the arrival times follow the exponential distribution. The transfer delay is thus:
\[ D = \frac{\rho E[T_p^2]}{2(1-\rho)E[T_p]} + E[T_p] + \frac{r(1-\rho/Q)}{2(1-\rho)} + \tau/2 \]  \hspace{0.5cm} \text{... (11)}

Where \( \lambda \) is the packet arrival time.

\( \rho \) is the load:

\[ \lambda E[T_p] \]

\( E[T_p] \) is the mean service time for the packet, both header and data.

\( r \) is the propagation time for the network.

\( Q \) is the number of active stations.

This model can account for single token network operation by setting the value of \( T_p \) to equal the ring rotation time, \( T_r \), when \( T_p \) is less than \( T_r \).

Equation (11) was used in figure 2.11 to describe the dependence of transfer delay on the load for a Token-ring network with the following parameter values:

1. Transmission speed = 1 Mbps.
2. Number of active stations = 30.
3. Transmission path = 800 m.
4. Frame length = fixed at 256 octets.
5. NIU latency = 1 bit time.

The Token-bus Transfer Delay Function:

The Token-ring model, shown in figure 2.10, may also be used to describe the Token-bus. The stations on the bus are placed in an order and serviced in a fixed rotation, similar to the Token-ring. In the Token-bus the switching delay, $\tau$, refers to the transmission of the entire token plus its propagation time to the next station, whereas $\tau$ for the token-ring refers only to the NIU latency (usually only one bit time long) plus the propagation time to the next station. Another exception is that the token is transmitted immediately after the completion of the last (allowed) frame in the Token-bus, so the value of $T_p$ is not adjusted for frames whose transmission time is less than their propagation time.

The transfer delay is thus described by equation (11), except that the value of $\tau$ is the sum of the token transmission time and the propagation time. This equation was also used to describe the function of load versus delay for the Token-bus network with:

1. Transmission speed = 1 Mbps.
2. Number of active stations = 30.
3. Transmission path = 800 m.
4. Frame length = fixed at 256 octets.

Figure 2.11 - Delay versus Load Characteristics.

The Token-bus network displays the greatest delay for these specific parameter values, irrespective of load. This is not generally the case as can be seen when studying the characteristics given by Acampora and Hluchyj [29]. They show that for a higher propagation delay to transmission time ratio (\(a\)) the CSMA/CD would perform significantly worse, for loads above 50%, whereas the Token-ring's delay would not increase to the same extent. The Token-bus will always give the greatest delay at low loads because of the long token rotation time, but for higher loads its delay is less than CSMA/CD's for large values of \(a\).
Conclusions with regard to delay by Bux [19] and Stuck [21] are more comprehensive:

1. The transmission rate influences the delay on the CSMA/CD network more than the delay on the other networks. For instance, in the CSMA/CD network, the delay tends to infinity close to a load of unity for a 1 Mbps network, whilst on a 10 Mbps network infinite delay would be experienced at a load of about 65%.

2. A change in packet length distribution, from a constant distribution to one that had large variations (according to a hyperexponential formula), indicated that the delay induced by three networks was largely unaffected by the packet length distribution.

3. A greater NIU latency increases the delay of the token-ring as the frame rotation is increased. This can also be seen by examining equation (11) in which the propagation delay, \( r \), is effected by the NIU latency and so effects the transfer delay. This change in transfer delay is multiplied by the number of active stations.
4. A change in network length affects the propagation time and so the delay times for all three networks. Changing the network length whilst keeping the other parameters fixed has the inverse effect of adjusting the transmission rate. Thus, as in 1 above, CSMA/CD is affected most by an increase in network length because the vulnerable period of the transmission, again, is increased. The two token networks are affected only in as much as the propagation delay affects the token rotation and frame transmission times.

An important conclusion reached by Murray and Enslow [25] and Stuck [21] is that the delays caused by the higher level protocol and the interface circuitry are the major time consumers and that, by comparison, the access method does not significantly influence the time taken for data transmission. The efficiency of the control software is therefore more important than the access method used.

2.3.2 Reliability

Two factors which often influence the choice of a network architecture are its resilience under error conditions and its ability to recover quickly from their
occurrence. Phinney and Jalatis [23] propose a number of error sources in their discussion of the error handling capabilities of the IEEE 802.4 bus. These are used here to examine the error handling abilities of the proposed architectures.

The error sources are:

1. Signalling and Noise Errors
2. Medium Failure
3. Transmitter and Receiver Failure
4. Repeater Failure
5. Station Malfunction
6. Station Failure

Signalling and noise errors are the result of environmental conditions such as impulse (terrestrial) noise or Rayleigh (shot) noise. These errors may also be a result of non-linear medium characteristics. Their occurrence would cause infrequent and intermittent errors on the network.

The medium may fail because of shorts or open circuits caused by, for example, moisture, bad connections or breakage.

Transmitter failure may manifest itself in two ways.
Firstly, the transmitter could continuously send data (jabber) or, secondly, the data rate or transmission power could drift. Receiver failure could manifest itself as deafness or signal attenuation. The frequency of occurrence of these errors is dependent on the design of the station. For instance, a jabber control is generally required so that a jabbering station does not cause total network failure for a prolonged period.

If a repeater node fails it may divide the network in two, allowing it to continue but as two separate networks. If the failure is in one direction only, part of the network will continue as a separate network whilst the other part will fail because it will hear network activity but will not be able to take part.

A station malfunction is due to some failure on the MAC layer, e.g. the station may detect errors in the frame check sequence or in the address when, in fact, they are correct.

Station failure could be caused by power failure or a higher-level command excluding it from the network.
CSMA/CD Error Handling:

The CSMA/CD network detects error conditions via the analogue collision detection hardware and the 32-bit frame check sequence (FCS).

It is unable to distinguish between short noise bursts and frame collisions, clearly a disadvantage, so there is no way of detecting short errors which trigger the collision detection hardware as errors and not collisions. This form of error will be recovered from by the frame collision re-transmission process. Should the noise error be too short to be seen as a frame collision, the corrupted bits will be detected by the FCS appended to the frame. A recovery procedure will then be initiated, usually leading to a frame re-transmission.

Medium failure will cause network failure as data cannot be transmitted under such conditions. Where jabbering occurs (case 3), the whole network will be disabled as the network will always be seen to be active, thus disallowing access to other stations. If the station fails in any other way (cases 5 and 6), only the offending station will be disabled.
Error Handling on a Token-bus:

Bit errors are detected by means of a 32-bit Frame Check Sequence (FCS).

An error of type 1, i.e. noise errors, will corrupt one or more bits in any frame. The FCS will detect the error and recovery is achieved by retransmitting the corrupted frame.

Should the medium fail, the whole network will be brought to a standstill as no data can be transmitted.

Where the transmitter malfunctions and begins to jabber, the network will fail until the jabber timer halts the failed station. In the other causes of station failure or malfunction, detailed above, the responsible station will be excluded from the logical ring at the next token pass. If the station fails while holding the token, the lost token error recovery procedure, given in the first part of this chapter, will be invoked.

The failure of a repeater will result in the isolation of a part of the network or the failure of the entire network, depending on the severity of the failure.
Error Handling on the Token-Ring:

Both Bux et al. [22] and Saltzer et al. [24] discuss the error recovery features of the Token-ring. In the Token-ring network any of the stations may be given the task of "monitor". It is the responsibility of the monitor station to maintain the network in a useable state and so it has the task of initialising most of the recovery procedures.

Signalling and noise errors can cause the loss of single frames, including the token. If the token is lost, the monitor's rotation timer will expire. It will then clear the network by transmitting a string of idle characters. Once they have been transmitted around the network and are received by the monitor, it will issue a free token and the network will continue as before. The corruption of any other frame will be detected by the FCS and a re-transmission will be requested by the destination node.

If either the link between any two nodes or one of the nodes fails, the network will be disabled unless some precaution is taken to isolate the disabled section. This is generally accomplished with the aid of isolation relay switches. The act of isolating the failed station or link
will result in the corruption of at least one frame, and perhaps the loss of a token. Recovery is as described above.

Two other conditions must also be considered in a Token-ring: a continuously circulating busy token and a duplicate token condition. The first condition could be caused by noise adjusting the free-token bit in the header, thus converting a free token into a busy one. The duplicate token condition may be caused by noise converting a busy token into a free one, which will allow a second station access to the network while the first is still transmitting. (The IEEE Token-Ring is a single token network.)

A circulating busy token is detected by the Monitor bit in the access control field of the frame header. This bit is always cleared by the originating station when it transmits a frame, and is set when a frame passes the monitor station. Should it be set in a frame received by the monitor node, meaning that the originating station has not reset it, the monitor will clear the ring by means of idle characters and then initiate the transmission of a free token.

A duplicate token error will be discovered by the token-holding station as the station will either read an
incorrect source address in a received frame, resulting from another node having begun its transmitting simultaneously, or it will receive a second token. In both cases the node will complete its frame transmission but will not issue a free token. After a period the monitor will discover that there is no token and perform a lost token recovery procedure.

2.3.3 Physical Level Considerations

The third aspect that influences the choice of an architecture is the physical constraints of the network, e.g. excessive cabling requirements and poor noise immunity. A description of the important physical considerations of the networks under scrutiny follows.

Physical Aspects of the CSMA/CD Bus:

With the listen-while-talk feature of CSMA/CD, the collision detection section of the interface unit must be able to sense the presence of a distant transmitter while being in close proximity to the local transmitter. This is achieved by sensing analogue levels on the network and comparing them to those levels being placed on the network by the transmitter. Because the difference in these levels can be small, sensitive analogue circuitry is required.
This analogue component is clearly a constraint.

Physical Aspects of the Token-bus:

There are a variety of signalling methods available to the IEEE 802.4 network designer, e.g. phase-continuous FSK and duobinary AM/PSK. In each case co-axial cable is used, making it prone to noise, and care must be taken to ensure that earth loops do not exist.

The sensitive analogue interfaces, required by CSMA/CD, are not required as the access method is entirely digital.

Physical Aspects of the Token-ring:

If a node or link on the Token-ring fails, the whole network will be brought to a standstill. Saltzer et al. [24] suggests that each link should be looped through a central hub. A relay, situated in the hub, can be activated remotely by the station. If either a link or the station itself should fail, the relay will bypass the station by shorting the failed station's input and output at the hub.
Ring synchronization: The ring is a closed loop and if there is no synchronization around the ring, data will be interpreted incorrectly. Clock recovery, and therefore synchronization, is achieved by synchronizing a local oscillator to the received data by detecting zero crossings. Owing to noise or imperfections in the clock recovery circuit, these zero crossings are distorted and cause deviations from the mean frequency in a random fashion, called "jitter". As the clock recovered from the received data is used to transmit the frame, the jitter can accumulate around the ring. In order to correctly track the data from the previous node, a narrow band Phase-Locked Loop (PLL) is required at each node. To maintain stability, the network should be broken at the monitor node to suppress the
accumulated jitter. The data is then received by means of a PLL and written to an elastic first-in-first-out (FIFO) buffer from where it is transmitted at a frequency fixed by a crystal oscillator.

2.4 Conclusions

This chapter has compared the IEEE 802 networks with reference to their performance, their reliability and their physical constraints. Of the IEEE 802 networks the Token-bus was chosen because:

1. It had the least complex physical level.

2. The difference in throughput and delay between the Token-bus and the other networks, for the specific requirements of this network, was not significant.

3. The Token-bus is at least as reliable as the other networks.

The introduction of the WD2840 provided two more reasons for choosing the token-bus, viz.:

1. It implements a version of the complex MAC layer,
thereby permitting a quick and simple network implementation and is also inexpensive and easy to use.

2. Single chip controllers for the CSMA/CD and Token-ring networks were not available.
References for Appendix J

27. WITTLIN, R.I., RATNER, D.V., "Choosing the Best Local Area Network for an Application", Computer Design, pp 143-149, Feb 1985
