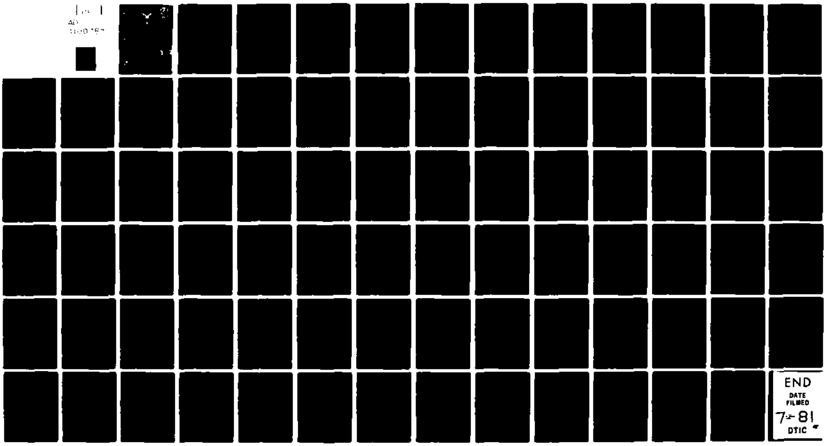
AD-A100 787 AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOO--ETC F/G 17/2
PROTOTYPE AND SOFTWARE DEVELOPMENT FOR UNIVERSAL NETWORK INTERF--ETC(U)
DEC 80 L R BAKER
UNCLASSIFIED AFIT/GCS/EE/80D-4 NL



UNITED STATES AIR FORCE

AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GCS/EE/80D-4

PROTOTYPE AND SOFTWARE DEVELOPMENT
FOR
UNIVERSAL NETWORK INTERFACE DEVICE

THESIS

AFIT/GCS/EE/80D-4

Lee R. Baker
Capt USAF

Approved for public release; distribution unlimited.

AFIT/GCS/EE/80D-4

PROTOTYPE AND SOFTWARE DEVELOPMENT

FOR

UNIVERSAL NETWORK INTERFACE DEVICE

THESIS

Presented to the Faculty of the School of Engineering

of the Air Force Institute of Technology

Air University

in Partial Fulfillment of the

Requirements for the Degree of

Master of Science

by

Lee R. Baker, B.S.E.E.

Capt USAF

Graduate Electrical Engineering

December 1980

Approved for public release; distribution unlimited.

Preface

The purpose of this investigation was to have an operational Universal Network Interface Device (UNID). This involved the complete testing of the prototype UNID along with the development and testing of software and required new hardware.

This investigation was limited to the development and testing of the prototype UNID and the software to support it. ~~There was~~ no attempt made to integrate the prototype into an actual computer communications network.

I would like to thank my thesis adviser, Dr. Gary Lamont, for his assistance and encouragement which was invaluable to the timely completion of this project. I would also like to thank my readers for their valuable comments and aid in this study. The high quality support of the lab technicians saved me much valuable time and I would like to thank Orville Wright, in particular, for his excellant help in the designing and procurement of the general purpose wire-wrap board. And finally, I wish to acknowledge my gratitude to my wife, Molly, for her encouragement and her effort in making this thesis more readable.

Contents

Page

List of Figures

List of Tables

Abstract

This thesis describes the third phase of the development and testing for a flexible message processor designed for computer communications network applications. This message processor is microcomputer based and is called a Universal Network Interface Device (UNID). The thesis describes the upgrading of the two system monitors, the design of two memory boards, the development of a test plan for testing the device, and the results of the testing that was performed. The device employs two microcomputer boards, a local card, a network card and two memory boards to provide the capability to interface local users to a computer communications network. The device also provides the capability to act as the node connecting two networks operating under dissimilar protocols.

PROTOTYPE AND SOFTWARE DEVELOPMENT FOR UNIVERSAL NETWORK INTERFACE DEVICE

I. Introduction

The purpose of this investigation was to develop an operational microcomputer based message processor with the inherent flexibility which would permit its utilization in different types of data communications network applications. The need for a universal network interface device was first proposed by the 1842 Electronic Engineering Group (EEG) of the Air Force Communication Service (AFCS) in an 1842 EEG/EEIC report, TR 78-5, entitled An Engineering Assessment Towards Economic, Feasible and Responsive Base-Level Communications Through the 1980's (Ref 1). There have been two previous investigations into the development of a universal network interface device, this investigation represents the third phase of the study effort towards an actual device.

The remaining sections of this chapter will address background information, the basic approach employed, the objectives of this investigation, and an overview of the subjects covered.

Background

In the past, telecommunication requirements on a typical Air Force base were satisfied in a rather simple manner by providing voice communication through telephone facilities plus a few low-speed teletypewriter and data circuits over base cable systems (Ref 1:2). However, with the recent tendency toward use of digital processors to accomplish base-level functions, the base-level telecommunication facilities needed to be reevaluated to insure they could support the increased data communications needs (Ref 1:2). An initial reevaluation was accomplished in the 1842 EEG/EEIC technical report TR 78-5 mentioned previously.

One facet of the TR 78-5 technical report involved the method of accomplishing the base-level message and data switching and distribution functions. At the base-level, distribution of data and other traffic is an important consideration since it encompasses user terminals and the communication paths connecting them into the local area network. There are, in general, numerous user terminals and thus costs associated with each terminal are multiplied by a large factor.

To satisfy the base-level message and data switching and distribution functions, the report postulated the need to connect any of the base digital processing devices to any terminal on the base and also the need to connect any base terminal to another terminal. To accomplish this interconnection, the "1842" report proposed three different types of network interconnections; the first was that of a star communication network with a centralized digital switch (Ref 1:162-163), the second was based upon the concepts used in the Advanced Research Project Agency network(Ref 1:164), and the last was that of the loop or ring network (Ref 1:164). The configuration the report finally recommended for the base-level data distribution network was a modification to the ring concept called a multi-ring network. This network consisted of a number of ring networks with a node providing interconnectivity between the rings. This multi-ring network concept, shown in Figure 1, offered particular advantages in terms of cost, evolutionary implementation and flexibility. A key to the multi-ring concept was the development of five different devices which could interface the multiple rings together.

Rome Air Development Center (RADC) was tasked with addressing the problem of the interface devices. It was recognized that the various interface devices proposed had similar features and were required to perform similar functions. Thus it appeared reasonable that a single flexible interface device could be developed which could meet the separate requirements of each of the five proposed interface devices of the multi-ring concept.

The above idea was incorporated into a Postdoctoral study program, and later became the basis of two research investigations. The first investigation (Ref 5) was the

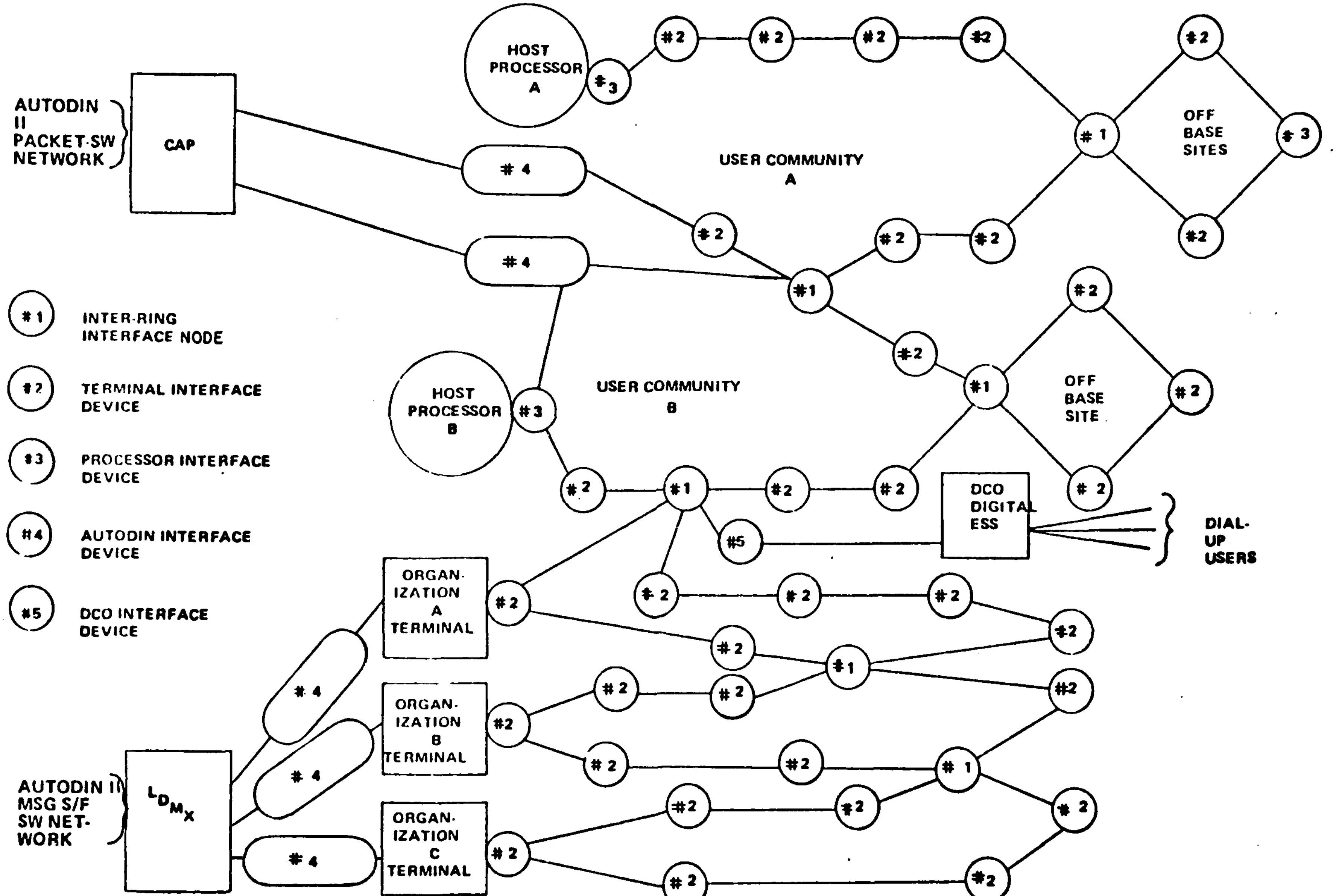


Fig. 1 Mult-Ring Base Network

first phase of the effort to develop the required message processor. The investigation involved definition of the functional requirements of the device, translation of these requirements into an overall system design, design of the hardware, and the development of the verification software routines. The second investigation (Ref 2) was the construction phase of the effort. The objective of this investigation was the implementation of a prototype Universal Network Interface Device (UNID). This included both the hardware and software required for device operation. The hardware implemention was partially completed and the software development was limited to simple testing routines.

Basic Design Approach

The basic design approach used in the first investigation involved two Z80 Microcomputer Boards (Ref 15) which shared a common block of memory. Each of the microcomputers would perform specific tasks within the framework of the device. To meet the "universal" qualification the UNID would have to interface to users directly or through Modems. Also, the device must be able to interface with a data communications network. To be able to perform these two types of interfacing, two special cards were designed, the Local Card and the Network Card.

The design, as specified in the first effort, was chosen to provide modularity of the hardware so that the device's capability could be increased by the addition of one or more Local or Network Cards. The Local Card has the capability of connecting up to four RS-232C users with the device. Through the addition of more Local Cards, additional users could be connected to the device. The Network Card has one full duplex serial port, which is used to receive a network message and determine whether the message is addressed to a UNID connected device. If the message is addressed to a UNID device, then the Network Card will place the message into the shared memory. The Local Card will perform the transfer to the correct device. If the message is not addressed to one of the UNID's devices, then it is passed along on the network. The Network Card will also take messages from shared memory, that have been placed there by the Local Card, and place these messages on the network communications lines. Two Network Cards would be used in the device if the UNID was to act as an inter-ring interface node.

The basic design called for a block of memory to be shared to allow communications between the two microcomputers used in the device. To meet this design feature, the Dual Processor Card was designed. This Card acted as a memory arbitrator between the two microcomputers when simultaneous accesses to the shared memory block were attempted. The Dual Processor Card also arbitrated the refresh signals developed by the two processors, so that the shared memory would be correctly refreshed and memory accesses would not interfere.

Thus the overall design of the device would be described as a dual processor device, with one shared memory partition to allow processor communication. One processor would handle the data flow to and from local users through the use of one or more Local Cards. The second processor would process the network traffic through use of the Network Card and the processor's capabilities.

The objective of the second effort was the implementation of a prototype Universal Network Interface Device. This included both the hardware and software required for device operation. The hardware implementation included the Local, Network, Dual Processor, and the two Microcomputer Cards. The software development was limited to simple routines used in testing various parts of the device.

At the time of the testing of the Dual Processor Card, during the second effort it was discovered that the Card does not provide perfect arbitration of the memory accesses. This testing showed that there are times when data bytes are incorrectly written to the stored memory, when the other processor is making memory accesses.

<u>Objectives of this Investigation</u>

The purpose of this investigation involved the complete testing of the prototype Universal Network Interface Device along with the development and testing of the software and new hardware required to have an operational device. Also involved was the modification of an existing monitor to improve the testing of both the hardware and software for the UNID. The following were the specific objectives of this project:

- Upgrading existing monitors
- Developing operating systems
- Developing needed hardware
- Testing

Upgrading Monitors. It was decided to upgrade the existing monitors because they had some nonessential features, such as the loading of programs from paper tape, and they also lacked some required features, such as the loading of programs from the disks of the Zilog MCZ 1/25 system. The monitors for both MCB's were upgraded to eliminate the unrequired and add the features required to load and debug UNID software.

Developing Operating Systems. The operating systems that were used in this project are modified versions of the operating systems developed during the first phase of the development of the UNID. These systems use techniques that are, for the most part, as simple as possible to accomplish the task. This was done to allow easier hardware/software isolation of any problems encountered during the testing phase.

Developing Needed Hardware. The main hardware that was developed was the system memory. To bring the system up to it's full memory capacity, required 32K of shared memory and 32K of dynamic memory for each side of the system.

Therefore two memory boards were developed, one with the shared memory and it's associated arbitration circuits, and one with enough memory to bring both sides up to their full capacity. Since the arbitration circuit was incorporated into the shared memory board, the Dual Processor Board has been eliminated.

Testing. In order to test the complete features of the universal network interface hardware design, both the Local and Network Cards needed to be included in the software effort. This required that an operating system be developed for each of the two processors. Within the different operating systems, certain techniques were used to accomplish a given task. For the most part, the techniques selected were the simplest to accomplish that task. This was to allow easier hardware/software isolation in the testing phase. While these simple techniques were adequate for testing, user application programs will require the employment of more sophisticated techniques.

Overview of the Thesis

This investigation involved the design of two memory boards to utilize the full memory capacity of the UNID along with the testing of the individual hardware modules that comprise the device. The software used to allow these modules to operate as an integral unit was also tested after the individual modules themselves were shown to be operational. Assembled versions of all system software are included in Appendix A. Circuit diagrams for all system hardware are provided in Appendix B.

The thesis is arranged into chapters which correlate roughly to the order in which the investigation was carried out. This Chapter serves as an introduction with the backqround information necessary to understand the thesis. Chapter II explains the modifications that were made to the monitor that was used during the course of the investigation. Chapter III gives a high-level description of the operating systems used with the UNID while Chapter IV discusses the hardware that was designed and partially realized. Chapter V explains the flow of the testing performed on the device. The thesis concludes with results and recommendations in Chapter VI.

The Appendicies referenced in this document are contained in a separate volume entitled, Prototype and Software Development for Universal Network Interface Device: Appendicies. This volume is on file in the Electical Engineering Department, at the Air Force Institute of Technology, Wright-Patterson Air Force Base, Ohio.

II. Monitor Upgrade

In order to be able to load, run, and test the software, and to test the hardware for the UNID, a system monitor was required. The Microcomputer Boards (MCB) used in the UNID come with a 1K MCB monitor in PROM (Ref 16). This monitor has some basic debugging commands and input/output routines. The input/output routines consist of a terminal handler and a paper tape reader/punch handler. The basic debugging commands, while adequate for debugging simple programs, were found to be very time consuming for a program of any significant length. Since source code (Ref 16) was available for this monitor, it was decided to use this code and upgrade the monitor as required to accomplish the debugging and testing for the UNID. This necessitated the removal of one unrequired function and the addition of those that were required.

The remainder of this chapter will describe the upgrading of the monitor and specify what sections were deleted, the commands and functions that were added, and the commands that were improved. The discussion will focus on the removed and added functions and justify the additions and deletions. A description of how the monitor operates, and how to use it's various functions, has been included in Appendix C, Monitor User's Manual.

Deletions

The major section of the monitor deleted was the paper tape reader/punch handler. There was no need for this section since it was unnecessary to read in or punch paper tapes with the UNID. The other deletions from the monitor were subroutines and other sections of the code that had to do with the operations of the main paper tape handling routines.

Additions

There were four commands/functions that were added to the monitor. These were the Load from disk, Fill memory, Move memory and Next (single step) commands. The Load function was required to bring programs into the UNID for testing and debugging. The Fill function was needed to fill memory with some known information and can also be used as a quick memory test. The Move command was added to make it possible to relocate sections of memory. This command also made it feasible to have a Load command for only one monitor, as will be explained. The Load command was added to enable programs to be transferred into the UNID memory space from another system's disk drives. The Next command was added to allow stepping through a program one instruction at a time. This is a great aid when debugging large complex programs!

Load Command. Since all of the software for the UNID was being developed with the Zilog MCZ 1/25 system, a command was added to the monitor to replace the paper tape commands. This command is used to read a file from the disks of the MCZ system into UNID memory. In order to transfer a file from the MCZ system into the UNID, a serial port on each system was required. On the UNID there were two serial ports available on the Local Card and two available on the Network Card. Since programs would have to be loaded into both sides of the UNID, either two different programs would have to be developed, one for each side of the UNID, or one program would have to be developed and then the ability to pass the program through shared memory to the other side of the UNID would have to be developed. It was decided to develop the load command for only one side of the UNID and pass the file through shared memory. This was done to enable the development of one program instead of two and it would require only one of the MCZ systems serial ports instead of two. It was decided to use the Local Card serial port as the port coming into the UNID. This was done because this card already had the RS232 serial port required to connect it to the MCZ system.

The load command required two programs, one for the UNID and one for the MCZ system. The program for the UNID requests the file name from the terminal and sends it to the MCZ system and then waits for the MCZ system to send back a header, which contains the length and load address of the file. After the header is received the program loads the file into system memory. The MCZ system program receives the file name sent by the UNID and checks to see if the file is on disk. If the file is on disk, the file header is extracted from the file and sent to the UNID followed by the file itself. If the file is not found, an error message is displayed on the MCZ console. The Load command has only been included in the monitor that is on the Local side of the UNID. The Network side has no use for a load command, since the files for the Network side are brought in through the Local side and then passed through shared memory to the Network side by use of the Move memory command.

Fill Memory Command. This command fills any specified section of continuous memory with a specified byte. This is done by requesting the low address, the high address and the byte to fill memory with from the console. Then the memory between the low address and the high address is filled with the requested byte and control is returned to the monitor.

Move Memory Command. This command moves any specified section of continuous memory from one address space to another address space. This is done in basically the same way as the Fill command by requesting the destination address and the source address from the console. With these inputs, the function then moves the memory section and returns control to the monitor.

Next Command. There was a need to be able to step through a program one instruction at a time when debugging a program. To fill this need, the Next, or single step, function was added to the monitor. This function steps through one program step (one Z80 opcode) each time the Next command is received by the monitor. After stepping through the program step the function returns control to the monitor and displays the contents of all of the registers on the console.

The Next function works by disabling interrupts then putting an enable interrupt instruction at the program counter minus one. An interrupt is then forced by use of the Counter Timer Chip (CTC) on the MCB board. This causes an interrupt to be pending because the interrupts have been disabled. The program is then started at the enable interrupt instruction, which causes one more instruction (the instruction to single step through) to be executed and the interrupt takes place. The Next interrupt handler routine then replaces the instruction, that was removed for the placement of the enable interrupt instruction, and displays all of the registers and returns control to the monitor.

Modified Functions

Two functions of the monitor were modified to make them more useful. These were the Display memory and display Registers functions. The Display memory function was modified to enable the memory to be displayed one location at a time and modified if required. The display Registers function was modified to enable it to display the contents of all of the registers simultaneously instead of consecutively.

Display Memory. This command originally was only able to display sections of specified memory all at once. If a memory location needed to be changed, it required the use of the Set memory command. The Display memory command has been modified to enable memory locations to be displayed one at a time. The contents may then be modified or not, and then the function moves on to the next continuous location or returns to the monitor. This is done by the function asking for a memory location and then displaying it and waiting until it receives the command to move to the next location, modify the present location or return to the monitor.

Display Registers. This function originally was a display Register (singular) function in that only one register could be displayed at one time. It has been modified so the either one register can be displayed at one time or all of the registers can be displayed. This is done by checking whether or not a register has been specified during the command. If it has not then all of the registers are displayed and the function will return to the monitor. If it has, then only that one register will be displayed and the function will wait until it receives the command to display the next register, modify the present register or return to the monitor.

Monitor Upgrade Summary

This chapter has covered the upgrading of the two monitors used to run the test software for the UNID. The use of the added commands and functions permits testing the operating systems and hardware in a much more efficient manner than was previously possible. The only real problem found with the modified monitors is with the Next command. While this command works very well, the problem is the fact that it uses the interrupt system to perform its function. This makes the testing of interrupt routines more difficult than required. Therefore, it is recommended that this function be changed in the future so as to not use interrupts. Assembly listings and detailed flow charts of both monitors can be found in Appendix D.

III. Operating Systems

This phase of the investigation involved the two software operating systems needed for the UNID. Because of their availablity, it was decided to use modified versions of the operating systems developed during the first phase of the development of the UNID. These operating systems are the simplest possible to test all of the parts and subsystems of the UNID. Most of the modifications that were done to the systems were to add routines that would enhance the testablity of the software and hardware of the system as a whole. These modifications will be explained in Chapter V which is on testing. Most of the remainder of the code changes were either to improve the speed of execution, to improve the readability of the program or to correct errors found during testing. Appendix A contains the complete commented listings of both operating systems. These listings contain detailed documentation which is sufficent for the complete understanding of the code. Also included in Appendix A are system flow charts to enable the reader to be able to understand the flow of the programs more easily. Because of the detailed documentation in Appendix A the rest of this chapter will provide a general overview of the structure for the main system software. The Local Operating System and its functions will be explained first followed by the Network Operating System and a brief summary.

Local Operating System

This operating system will be used to control the local side of the Universal Network Interface Device. A high-level flow chart of the operating system is shown in Figure 2. This flow chart will be used to explain the main operating system functions. The flow chart shows the main loop of the operating system but does not show any of the subroutines used in the system or any of the details of the operation of the system. The subroutine flow charts and the more detailed flow charts are included in Appendix A. A detailed design of this operating system can be found in Ref 5.

The first operation performed by the operating system is that of initialization. This includes the setting of certain processor registers, the initialization of the I/O ports and certain components on the processor card, and the initialization of the queues and tables needed for system operation. After the initialization is completed the main loop of the program starts executing. This loop consists of checking both of the multibuffer areas, these are the areas where the addresses of a sequence of frames are stored, to see if all frames of a message have been received. Next the Local Transmit and Local Busy queues are checked to see if either contains the address of a message. If an address is present then the message it points to will need to be transmitted to the appropriate local device. If either of the multibuffer areas indicate all frames have been

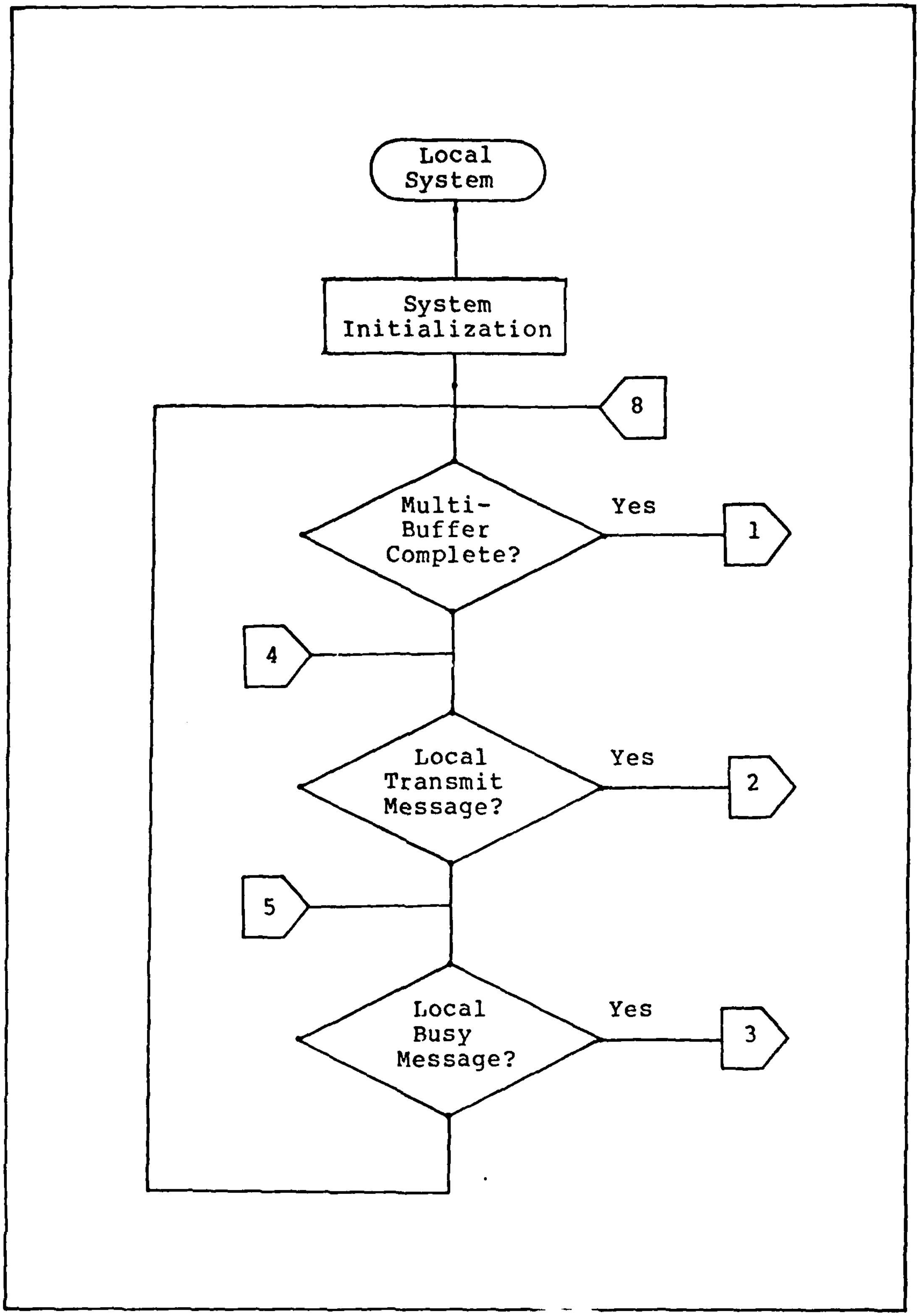


Fig. 2 Local Operating System Flowchart

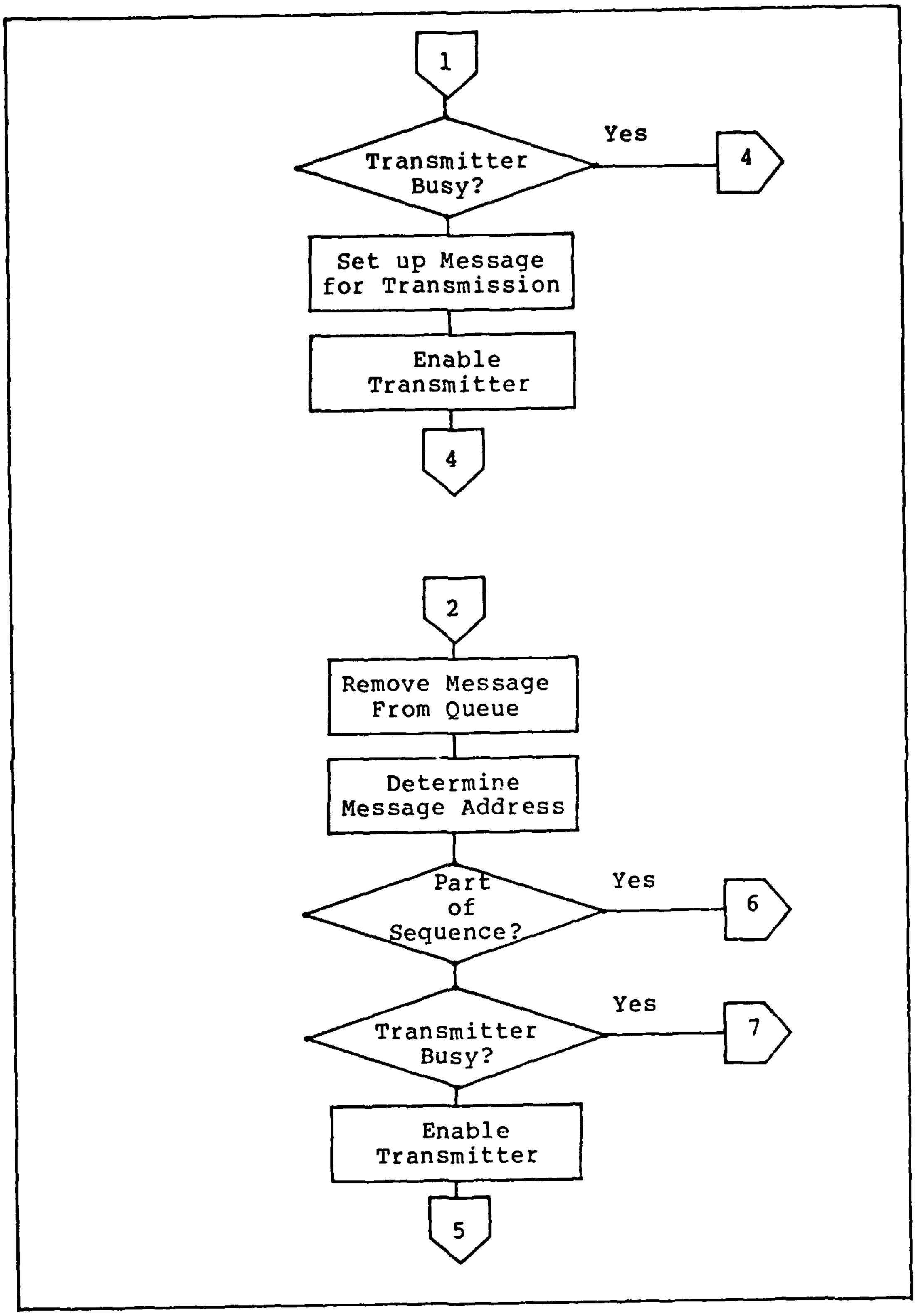


Fig. 2 continued

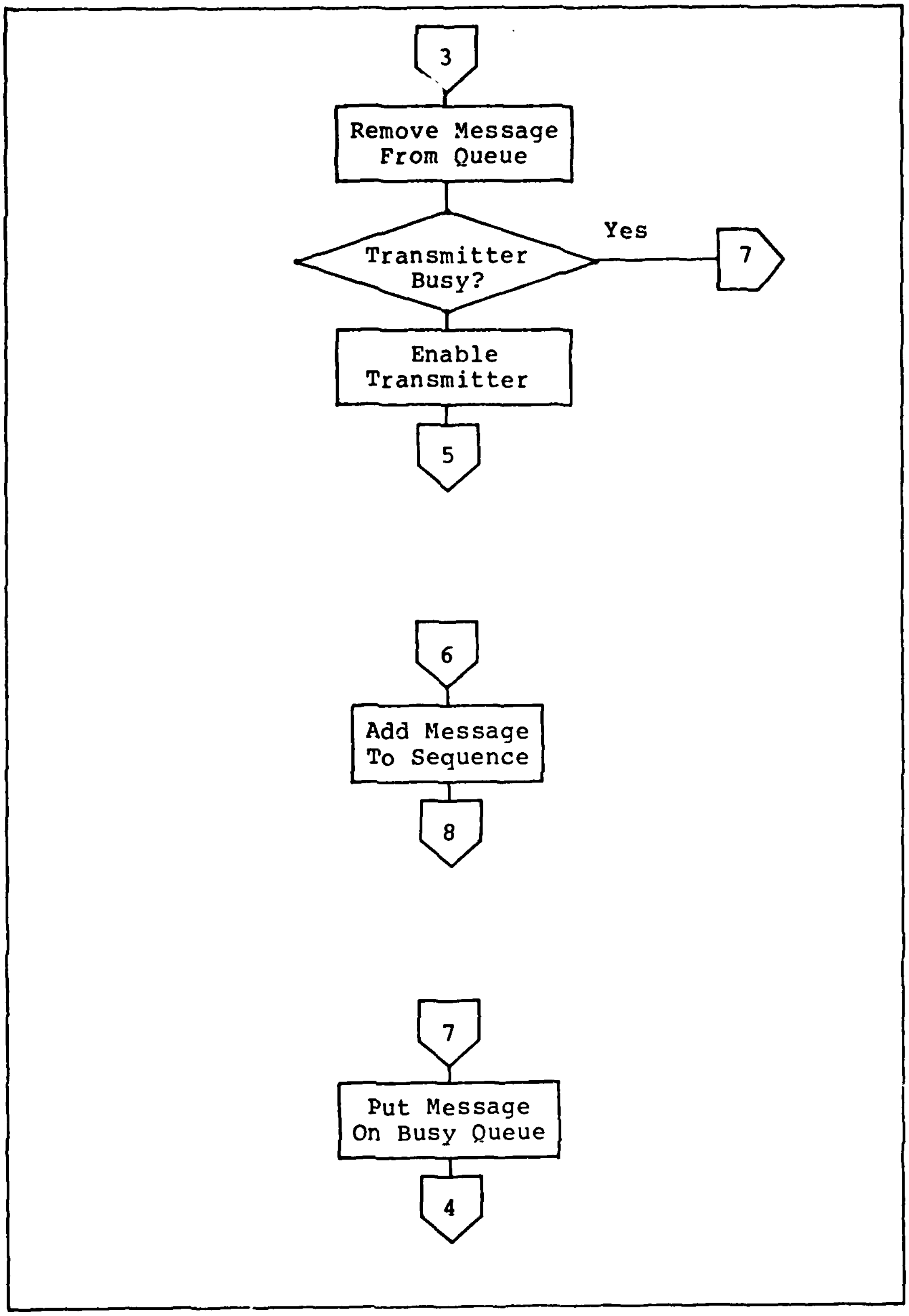


Fig. 2 continue

received, or one of the queues has a message address then a branch is made from the main loop to handle the indicated condition.

If one of the multibuffers indicates that all frames have been received, then a check is made to see if the transmitter is busy transmitting a previously recieved message. If busy then a return is made to the main program loop. If the transmitter is not busy then the address of the first frame in the sequence is loaded into the transmitter buffer area. After this the transmitter is enabled and a return is made to the main loop. Once the transmitter is enabled it transmits all of the frames in the sequence to a local device by interrupts. Anytime the transmitter buffer is empty an interrupt is generated and another character is loaded into the buffer. This continues until the entire message has been transmitted.

When a message address is found on the local transmit queue it is removed from the queue and the message destination address is checked to see which local device the message is intended for. After the local device is determined the program checks to see if the message is part of a sequence. If the program determines that the message is part of a sequence then the address of the message is put in the multibuffer area for that sequence and a return is made to the main program loop. In the case that the message is not part of a sequence, the transmitter is checked to see if it is busy transmitting another message. If the transmitter is busy then the message address is put on the Local Busy queue and a return is made to the main loop. A transmitter that is not busy will cause the program to put the address of the message into the transmitter buffer area and enable the transmitter, followed by a return to the main loop.

If a message address is found on the Local Busy queue then the address is removed from the queue and a check is made to see if the transmitter is busy. After a determination is made as to the status of the transmitter the flow of program control is the same as that explained above for the Local queue. That is, the frame is transmitted if the transmitter is not busy or else the frame is returned to the Local Busy queue if the transmitter is busy. In either case a return is made back to the main program loop.

This concludes the discussion of the Local Operating System. The remainder of this chapter will be used to explain the Network Operating System.

Network Operating System

This operating system controls the network side of the Universal Network Interface Device. Figure 3 is a high-level flow chart of this operating system. As with the Local Operating System, this flow chart will be used to explain the main functions of the operating system and does not depict subroutines or details of system operation. The flowcharts for these are included in Appendix A along with

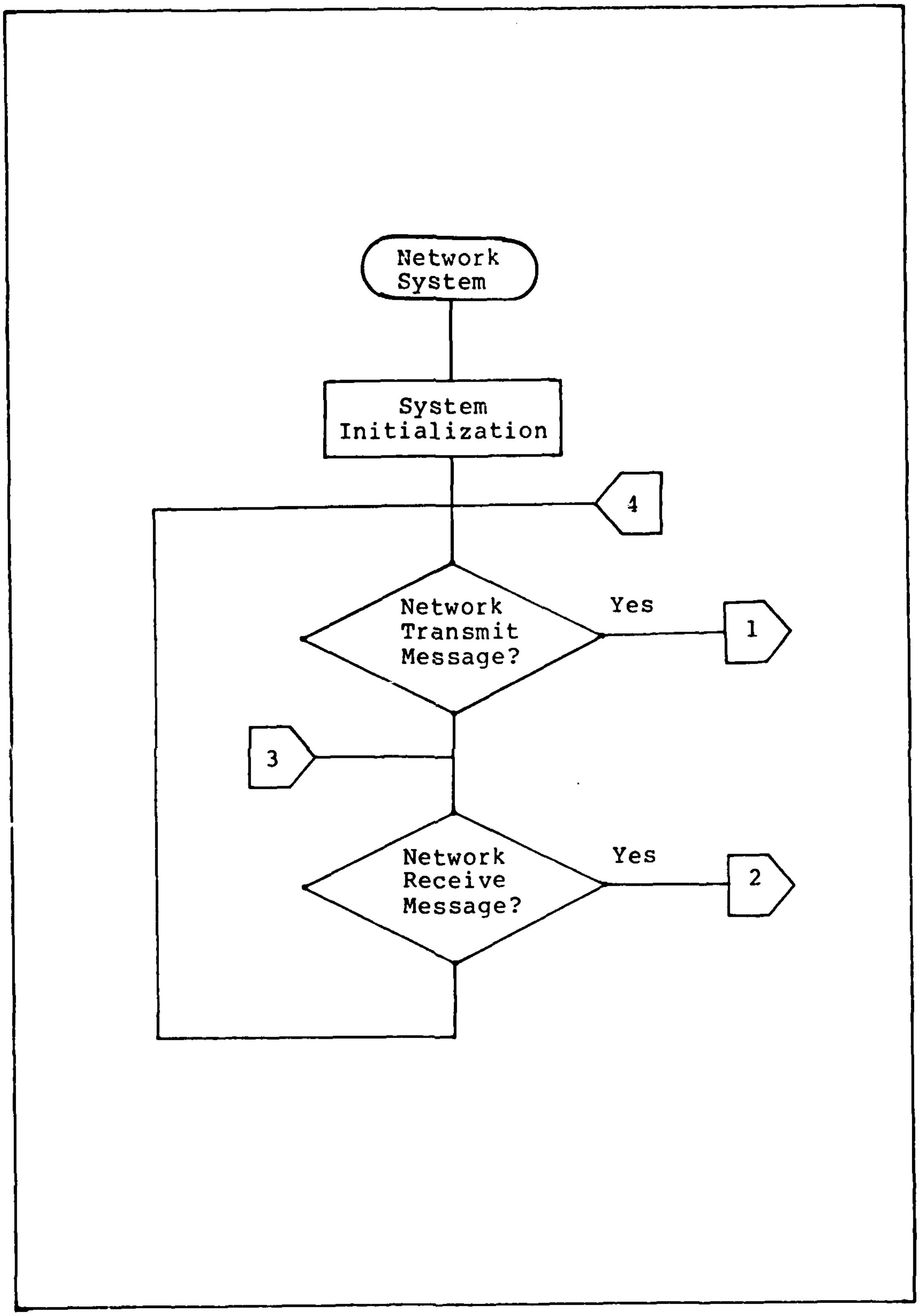


Fig. 3 Network Operating System Flowchart

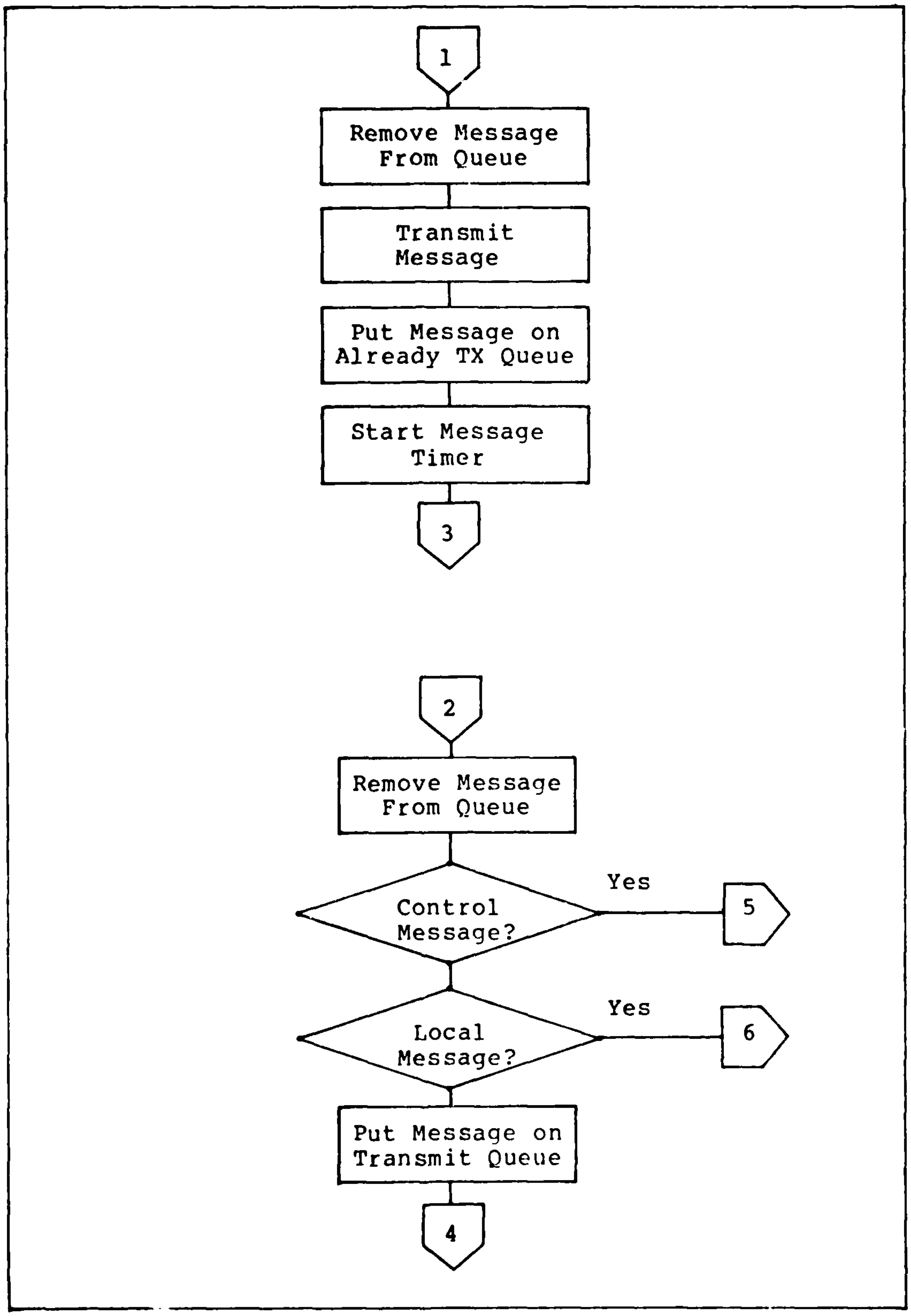


Fig. 3 continued

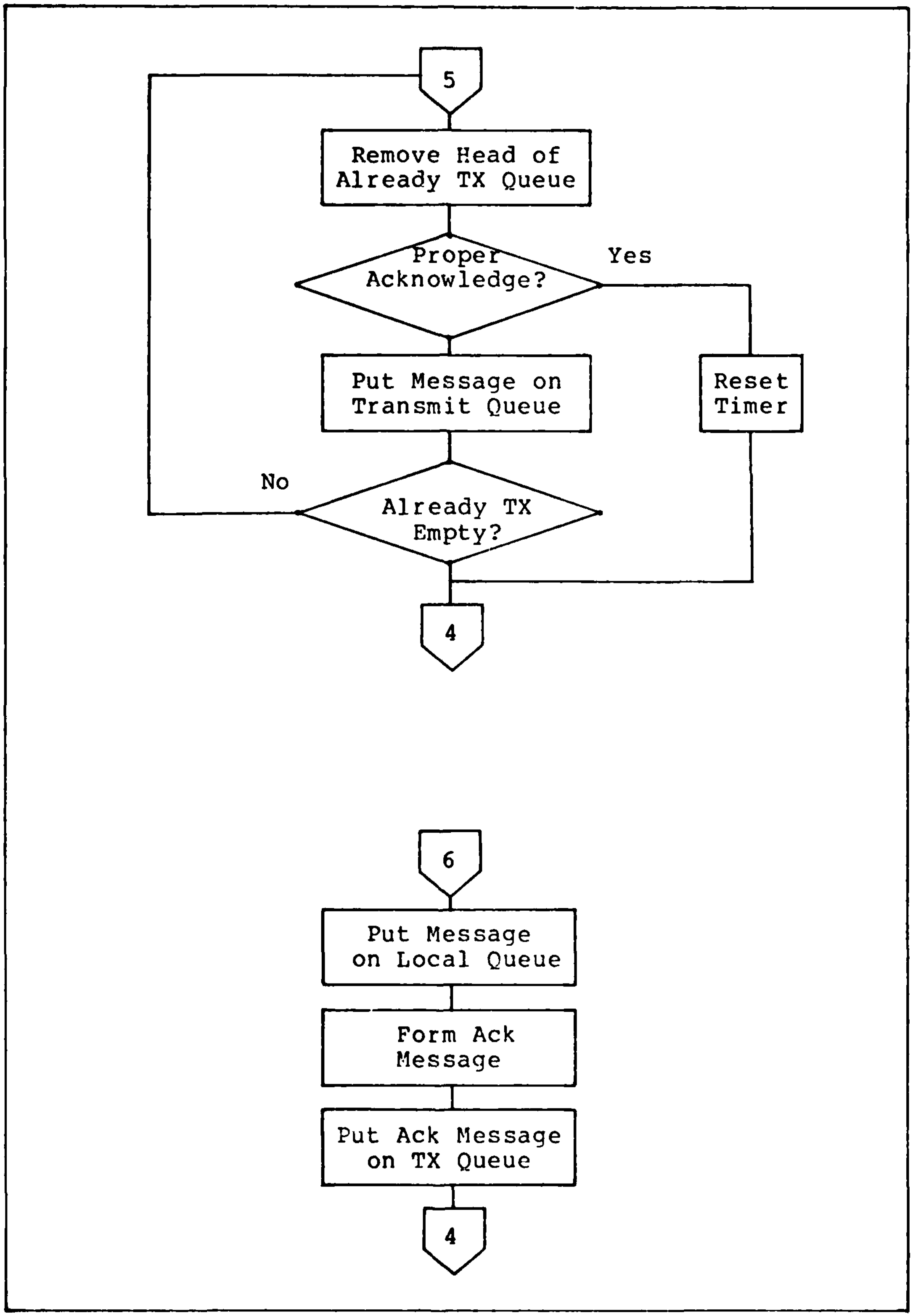


Fig. 3 continued

the listings for the operating system.

Like the Local Operating System, the first operation performed is that of initialization. This includes the setting of some processor registers and the initialization of the queues and tables needed for operation. After the initialization is complete the main program loop is entered.

The main loop checks both the Network Transmit and the Network Receive queues for a message address. If a message address is found in either the Transmit or Receive queues then a branch is made out of the main program loop to handle the message. In the case that a message address is found on the Network Transmit queue, the message address is removed from the queue. After this the message is transmited to the network by the transmit subroutine. Upon return from this subroutine the message address is put on the Network Already Transmitted queue. The network message timer is now started. This timer is used to time out the amount of time after a message is sent until an acknowledgement for the message must be received. If the acknowledgement to the message is not received within that time frame, the message is automatically assumed to have been received incorrectly and is retransmitted. After the timer is started the program returns to the main loop.

When a message address is found on the network receive queue, the address is removed from the queue and the message is checked to see if it is a control message (an acknowledge message). If it is a control message then the message address at the head of the Network Already Transmitted queue is removed. This message is then compared with the acknowledge message to see if this is the message that is being acknowledged. If it is the correct message then a return is made to the main program loop. If not the correct message, then the message address is put back on the Network Transmit queue and the next message address is removed from the Already Transmitted queue and the comparing process is repeated. In the case where it is determined that the message was not a control message, the message is checked to determine if it is for a device connected to the UNID's local side. If it is for a UNID connected device, then the message is put on the Local Transmit queue. After this an acknowledge message is formed and put on the Network Transmit queue. In either of the above cases, after the message, either the acknowledge message or the orginal message, is put on the transmit queue a return is made to the main program loop.

Operating Systems Summary

This chapter has given a general overview of how the software for the UNID will operate. This software in and of itself is not sufficient to be used in an operational UNID but can provide a good framework on which to build operational software. The code, as developed, allowed one port of the transmit section of the Network Card to be looped to the receive section of the same port. This allowed all parts and subsystems of the UNID to be tested. It is recommended that during a future investigation involving the UNID, that an attempt be made to code the operating systems in a high-order language, like Zilog PLZ (Ref 8), as this would enhance the readability of the software as well as make it easier to modify and maintain.

IV. Hardware Systems

During this phase of the investigation two memory boards were designed. One of the memory boards was called a Shared Memory Board and the other was called a System Memory Board. The Shared Memory Board was designed because of problems that were being encountered with the present Dual Processor Card. The main problem being encountered was that it was not arbitrating memory accesses properly with the Z80 refresh signals when both processors were making rapid alternate accesses to the memory. Because of this problem it was decided to design a new memory board with a new arbitration circuit using static instead of dynamic memory to try to eliminate the arbitration problem. Also designed was a System Memory Board that would bring the non-shared system memory of each processor up to its full capacity. This memory board was designed because of the anticipated need for full memory capacity in an operational Universal Network Interface Device. With the need for these two new boards it was found that the present wire-wrap board, that had been used in the construction of the UNID, did not have a sufficient number of IC (integrated circuit) sockets for these new boards. Because of this a new general purpose wire-wrap board was designed.

During the testing of the Local and Network cards of the UNID it was found that a new reset circuit would have to be developed. The present reset circuit was causing a loss of memory information if the reset button was held down for more than a very short time. Because of this problem a new reset circuit was designed that would eliminate the problem.

The remainder of this chapter explains the design of the new wire-wrap board, the design of the new memory boards, and the new reset circuit design.

Wire-Wrap Board

When it was determined that two memory boards would have to be designed and built for the UNID, it was discovered that the type of wire-wrap board that had been used in the construction of the previous boards was not very general in nature. It also had a very limited capacity and range of IC sockets. There are only enough sockets on the board for sixty 14 or 16 pin IC's and for two 40 pin IC's. If no 40 pin IC's are used then seven more 14 pin IC's or six more 16 pin IC's could be used on the board. Since the number of IC's needed for each memory board exceeded the maximum that could be put on the present wire-wrap board, it was decided to design a new more general purpose wire-wrap board that would suit the needs of the UNID much better.

The board that was designed has 21 parallel rows of holes with 68 holes in each row. This gives the capability of putting eighty 14 pin IC's or seventy 16 pin IC's on the board. There are also power and ground busses running up either side of the board and a 122 pin edge connector on the bottom. The board has been designed with three extra rows of holes at the top of the board with spacing that will allow any IC presently marketed to be mounted on the board.

It is hoped that this board is sufficiently general in nature to make it possible to implement any circuit needed in the future for the UNID on the board.

Shared Memory Board

The original design of the Universal Network Interface Device called for a block of shared memory and a Dual Processor Card to arbitrate the accesses to this memory. In this design the memory was dynamic. This caused the arbitration circuit to be quite complicated because of the need to refresh the memory and to keep this refreshing from interfering with the normal memory accesses. The circuit that was designed and implemented did not arbitrate the memory accesses correctly when both processors were trying to access the memory alternately and repeatedly. This caused a loss of information that was stored in the memory at certain times. It was decided to use static memory to solve this problem. Since static memory requires no refreshing, this reduced the complexity of the arbitration circuit at the same time that it eliminated any conflicts since there was no need to refresh.

The arbitration circuit used was a circuit that was in an article in the July 1978 issue of BYTE magazine (Ref 4). This arbitration circuit was used because it was perfectly tailored to the needs of the UNID shared memory concept and it was also relatively simple in comparison to the circuit used on the Dual Processor Card. Because of this simplicity and the small number of ICs needed to implement this arbitration circuit, it was decided to eliminate the Dual Processor Card and to include the arbitration circuit on the Shared Memory board itself. The remainder of this section will describe the arbitration circuit, the theory behind it and how it operates.

Basic Design Concept. The design of the UNID consists mainly of two Z80 microprocessors operating independently of each other, each supported by 32K bytes of memory. The processors are indirectly linked to each other by a shared 32K memory block. The shared memory is addressable by either processor as the upper 32K of it's 64K of addressable memory space. This shared memory has its own address and data busses which are gated to the two processors' busses by the arbitration circuit. This circuit also controls the signals coming from each processor when (1) either processor performs a memory read or write operation or (2) either processor does an op code fetch from the shared memory, or (3) machine instructions combine (1) and (2).

Z80 Memory Reference. In a two-processor environment, the basic problem was to allow both processors access to the same data, at the same point in time, in the least amount of time. There had to be some way to delay one processor's memory request until the other processor's request was completed. In the Z80 case, there are two basic instruction cycles which effect memory. The first is an instruction fetch cycle (M1) which normally requires 4 clock cycles.

During this machine fetch cycle, the first half reads the memory word addressed by the program counter while the second half generates a refresh address for any dynamic memory being used. The other machine cycle (M2), data read or write to memory, requires 3 clock cycles. Each of these machine cycles can be extended through use of the Z80 wait input. During the second clock cycle of the different machine cycles, the Z80 checks its wait input to determine if a wait state is requested. If so, an additional clock cycle is added to the executing machine cycle and the wait input is again checked during the middle of this clock cycle. This checking and wait generation continues until the wait request is removed (Ref 5).

Basis of Design. The two basic machine cycles and the wait input capability provided a method to arbitrate shared memory references. At any given point in time, one processor could be in seven different states with regard to a memory reference. These seven states equate to the different cycles involved in the two basic memory reference cycles. For the second processor to access this same memory requires it to be in cycle one of a machine fetch cycle or cycle one of a machine memory read/write cycle. It has been shown (Ref 4) that if one processor's clock is 180 degrees out of phase with the other processor's clock that the processors could operate in parallel with minimum reduction of processor speeds. This required that the memory being used be static and have a memory cycle time less than the processor's clock period.

For the Z80, with a clock frequency of 2.5 megahertz, this equated to memory with an access time of 300 nanoseconds (ns) and for a Z80A, with a clock frequency of 4.0 megahertz, it required an access time of 200 ns. In anticipation of an upgrade to the faster Z80A processor, the memory used was static with an access time of 150 ns.

Memory Design. A block diagram of the Shared Memory Board is shown in Figure 4. The heart of this diagram is the shared memory arbiter which selects which processor will be connected to the shared memory address and data busses if a conflict occurs. The arbiter will be discussed later in this section. The other blocks of the diagram are the address bus selector, data bus buffers, shared memory block, column decoder and the chip select MUX. The address bus selector, which is a block of four 2 to 1 data selectors, determines which processor's address bus will be connected to the shared address bus. The data bus buffers perform the same function for the data bus, while also determining whether data will flow into or out of the shared memory bus. The shared memory block is an array of Intel 2114 Static RAMs (Random Access Memory) arranged in 8 columns by 8 rows to give 32K bytes of continuous memory. The column decoder uses address lines A13 and A14 to select two of these columns while the chip select MUX uses address lines A10 thru A12 to select the particular row of chips within these columns. The signals to control both the data bus

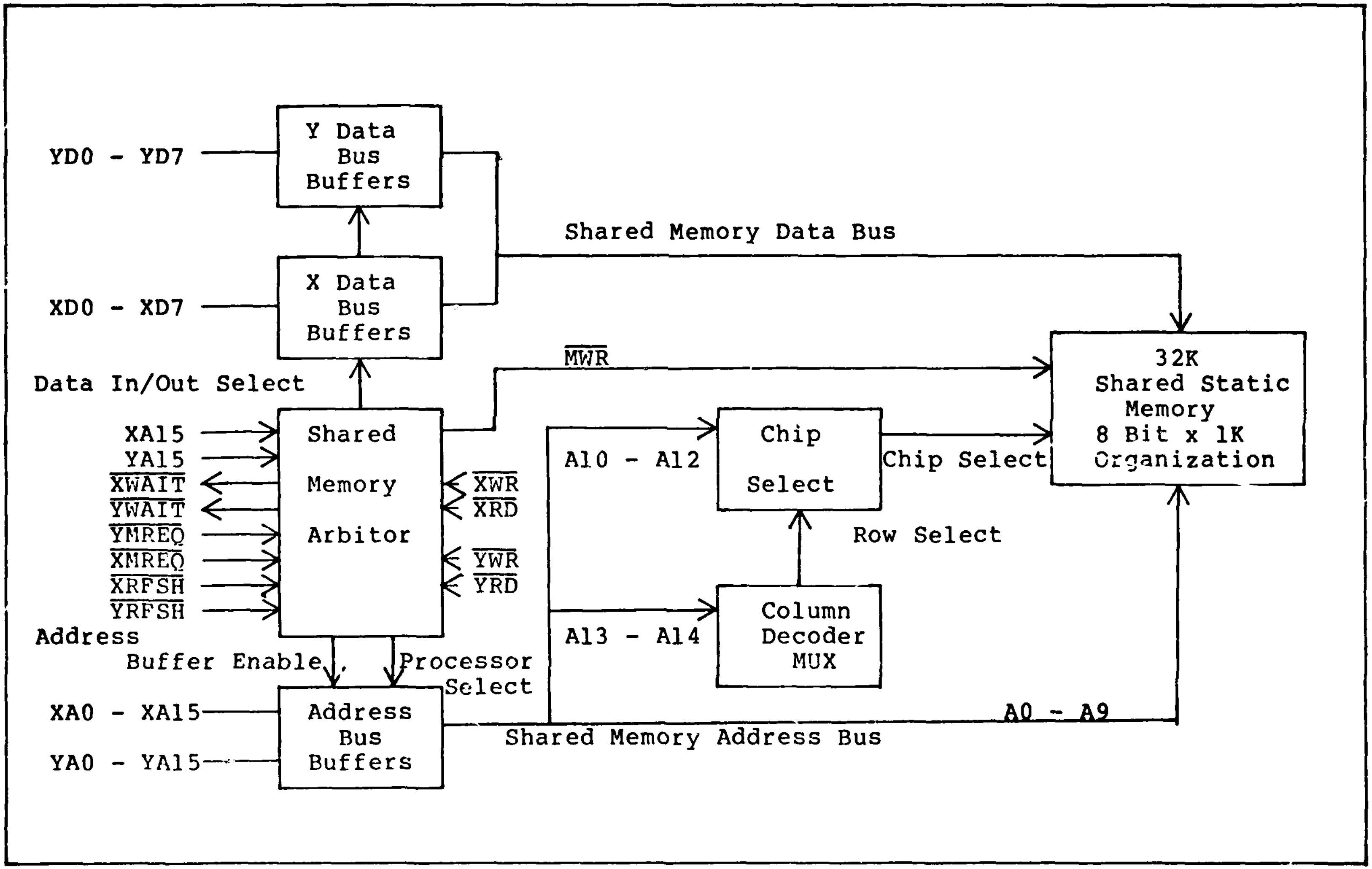


Fig. 4 Shared Memory Block Diagram

buffers and the address bus selector are generated by the arbiter.

Arbitor Logic. A logic diagram for the arbiter is shown in Figure 5. Each processor provides signals to the arbiter which identify a valid shared memory access request. The two NAND gates in Figure 5 receive RFSH, MREQ, and A15 (the high order address bit signal) from their respective processors. MREQ indicates that a memory read or write operation is underway: either A15 line going high identifies the shared memory as the object of the request; and the RFSH lines insure that the dynamic memory refresh strobe from one processor will not interfere with the shared memory access request of the other. The two tri-state buffers provide an opposing grant or deny shared memory bus access proviso that is strictly first come first served. A request from the local processor will cause the the top tri-state buffer to drive XSELECT low, and at the same time disable the bottom tri-state buffer. The network processor will be locked out for the length of the local processor's memory request. Now suppose that the network processor does make a request for bus access when the local processor is using the bus. This condition will force the OR gate that is connected to the two NAND gates to its low state, activating the YWAIT line. The wait signal will continue until the local processor concludes its memory access. Under no circumstances, however, will the network processor be forced into more than one wait state for this local processor access. When XMREQ

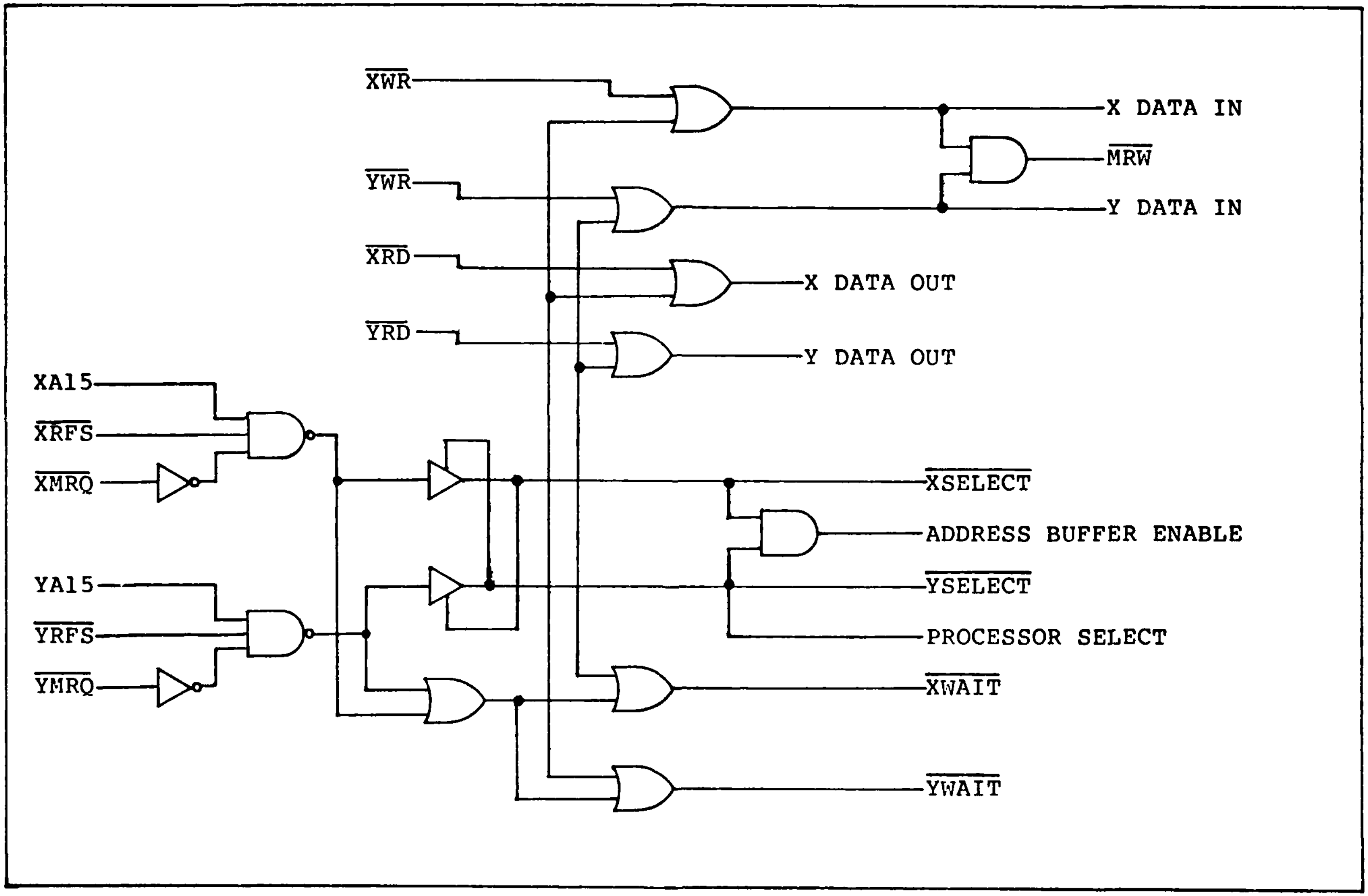


Fig. 5 Arbitor Circuit

goes high, XSELECT follows suit, enabling the bottom NAND gate and granting the network processor request.

A complete schematic of this board is included in Appendix B. The Shared Memory Board was implemented on the general purpose wire-wrap board that was developed for use with the UNID.

System Memory

The Zilog MCB microcomputer boards used in the design of the Universal Network Interface Device have provisions on board for 4K bytes of 2708 EPROMs (Eraseable Programable Read Only Memory) and 4K bytes of dynamic RAM memory. It was anticipated that the EPROM memory will contain the operating system for the UNID and that the dynamic RAM memory will be used for storing variables and tables needed by the operating system. The memory on the MCB board, although sufficent for a simple test operating system, is not large enough for the operating system that will be required for an operational UNID. Because of this need, another board was designed that has enough memory to bring each processor to 32K bytes of non-shared memory. It was also anticipated that the 4K bytes of EPROM space provide on each MCB board would not be sufficent to hold the entire operating system of an operational UNID. To handle this, the memory on the new board, from 4K to 16K, was divided into three banks of memory with 4K bytes of dynamic RAM in each bank. If more EPROM is required one of these banks can be removed and EPROM put into its address space. This will provide up to a maximum of 16K bytes of EPROM which should be enough for even the most sophisticated operating system. The remainder of this section will describe the design and general operation of this memory board.

Since the circuitry for the memory board for either side of the UNID is the same, a general block diagram that can be applied to either side will be shown. This block diagram Figure 6, shows buffers for all signals coming on the board, two page decoders, two memory address MUXs, one bank of dynamic memory using 4K x 1 dynamic RAMs and one bank using 16K x 1 dynamic RAMs. The page decoder uses address lines A12 thru A15 to decode which 4K memory page is being accessed in the case of the 4K x 1 RAMs and to decode when the addresses from 16K to 32K are being accessed in the case of the 16K RAMs. These signals along with address lines A0 thru A11 are used as inputs to the memory address MUXs to generate the CAS (column address strobe), RAS (row address strobe) and the address signals needed by the memory. In the case of the 4K x 1 RAMs, addresses A0 thru A5 are sent to the memory during CAS and addresses A6 thru A11 during RAS. For the 16K x 1 RAMs addresses A0 thru A6 are gated during CAS and A7 thru A13 during RAS.

A complete schematic of this board is included in Appendix B which show the circuits for both sides of the UNID. These circuits, though on the same board, are completely independent from each other. The System Memory Board was implemented on the general purpose wire-wrap board

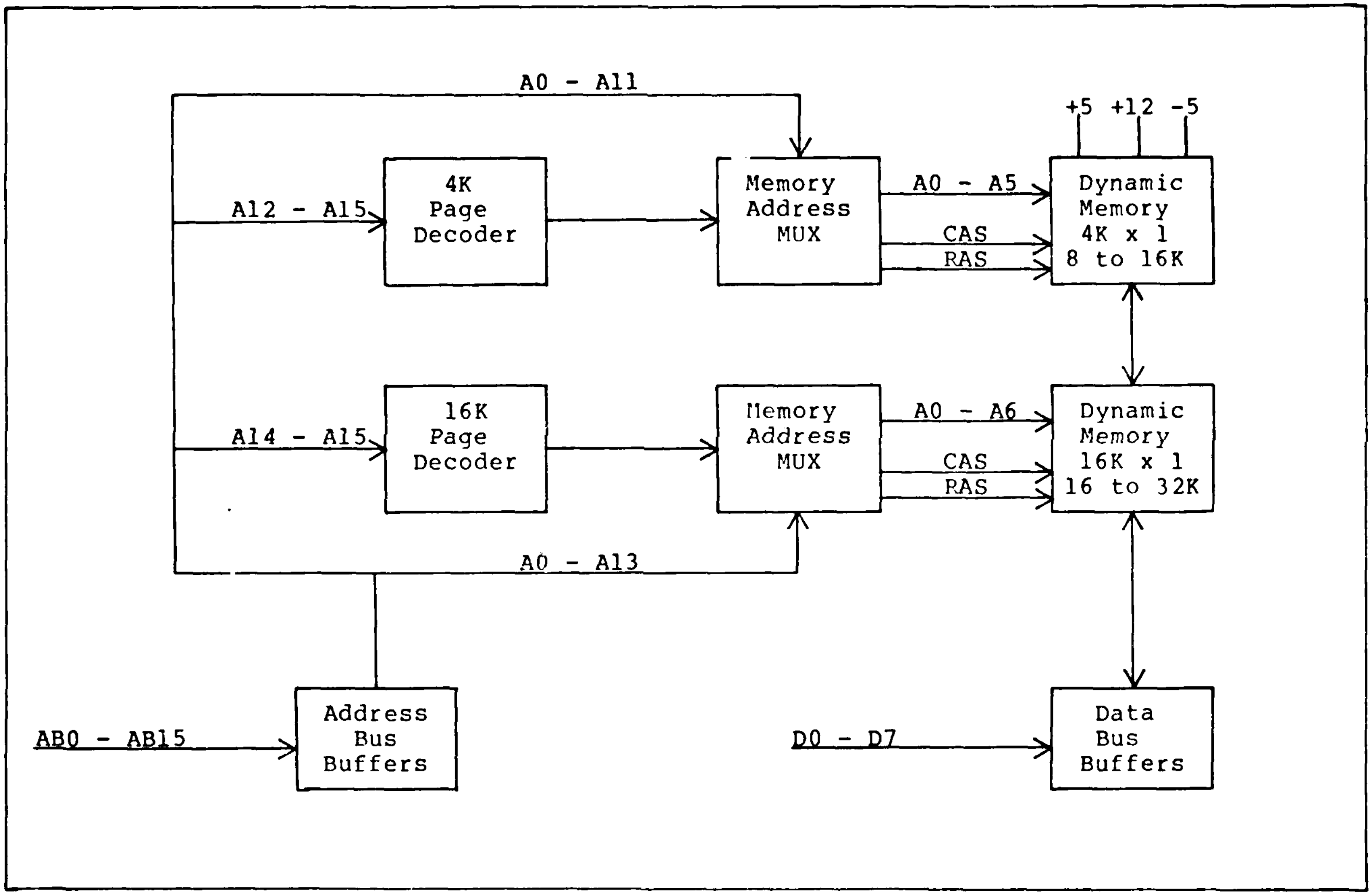


Fig. 6 System Memory Block Diagram

that was developed during this investigation.

Reset Circuit

A reset circuit was developed to reset both processors of the UNID independently. This circuit was developed because of problems that had been encountered with the previous method of reseting the processors. Previously the reset lines coming from the MCB boards were pulled to ground by the use of a push button switch that was not debounced. The problem encountered with this was that if the switch was in contact with ground for more than about 2 milliseconds the dynamic memory would not be refreshed and would lose information. Since the design of the monitor required a reset relatively frequently and this lead to the dynamic memory losing information, if the reset button wasn't released soon enough it was decided to design a new reset circuit.

The circuit that was designed is shown in Figure 7. Two push button non-debounced switches were used in the circuit and were each debounced using two NAND gates for each switch. The output from these now debounced switches was input into two monostable multivibrators, one for each switch. The time constant of these multivibrators was set to give a 70 us negative pulse. This signal is ANDed with the power-on reset pulse which is developed by a resistor, capacitor and diode as shown in Figure 7. The output of these AND gates are then input to the reset lines of each MCB board.

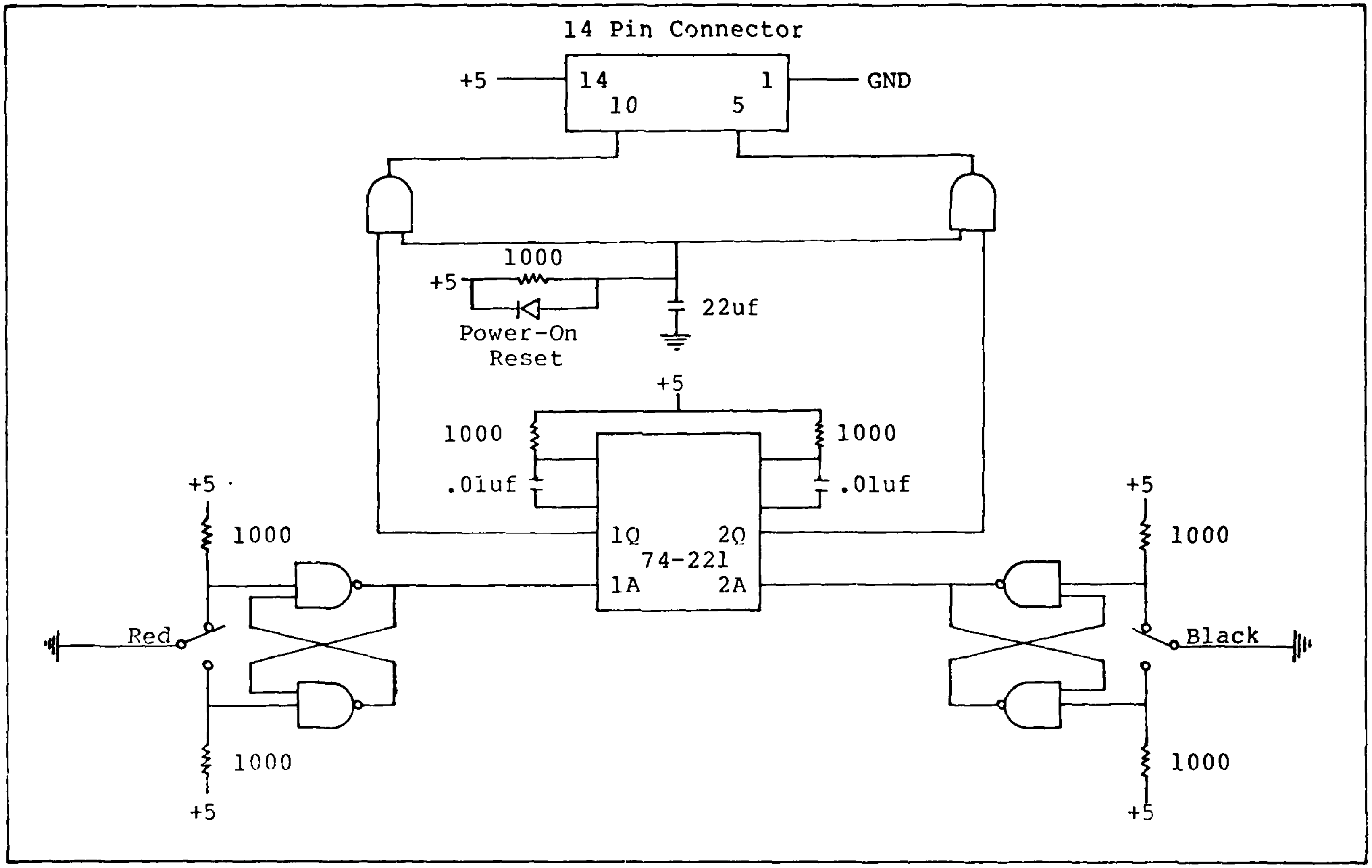


Fig. 7 Reset Circuit

This circuit was implemented on a piece of perf-board with wire-wrap sockets. The circuits outputs and the power and ground connections are connected by a length of flat ribbon cable which is attached to connector J20 on the motherboard of the UNID. A schematic of this circuit has been included in Appendix B.

Hardware Systems Summary

This chapter explained the design of the general purpose wire-wrap board, both the Shared Memory and the System Memory boards and the reset circuit. At the present time the Shared Memory Board has been implemented and tested with 8K bytes of memory. Testing of the board indicates that all of the problems associated with the Dual Processor Card have been eliminated. The arbitration between the two processors appears to be perfect and no loss of information has been experienced. The System Memory Board has at this time been fully designed but has only been partially wire-wrapped. The reset circuit that was developed has solved the problem of memory information being lost because of the reset button being held down too long. With the design of the above hardware systems completed, it is hoped that the next investigation involving the UNID can concentrate more fully on the software needed to make an operational system feasible.

V. Testing

The last phase of this investigation involved the complete system testing of the Universal Network Interface Device. This testing was done in an incremental modular fashion. It was decided to do the testing by modules because the design is easily broken into functional modules that can be tested as individual units with simple test procedures. The incremental approach was chosen because it has obvious advantages in that it is self-focusing on the source of errors. Any new problem is caused either by a defect in the most recently added module or by some new interaction between the new module and the rest of the system. This makes the isolation of the module that is causing the problem a relatively easy task. The testing of the individual modules was done from the bottom module up to the top module (bottom-up). Bottom-up testing was used because it focused first on the important lower modules and it reduced the amount of code needed by eliminating the need to write stubs. It also was a much more logical way of testing the hardware since the boards and cards were made up of easily defined modules that could be individually tested. The reduction in code for the software was realized because of the fact that no drivers would have to be written when using bottom-up testing. The drivers that would normally be required could be simulated by use of the monitor with it's ability to set the value of any processor register. Top-down testing would still have required stubs because these could not have been simulated with the monitor. The remainder of this chapter will explain the test plan that was used to test the UNID and the results obtained from the tests that were performed. The test plan will be presented first followed by the test results and a brief summary.

Test Plan

This section discusses the plan employed in testing the hardware and software of the Universal Network Interface Device. The test plan used is an incremental modular bottom-up test plan. An incremental modular bottom-up test plan takes the design as a whole and breaks it up into small functional modules and then tests these individual modules. This means that the first tests are performed on the bottom modules, in the case of hardware, for example, a single IC or a small functional group of ICs, is tested and then the tested modules are put together to form a larger module and tested again. After tests on the larger modules are complete then these modules are joined and tested. This process continues until the entire design has been tested.

In the case of the UNID, the design was broken down into modules as shown in Figure 8. The design was first separated into hardware and software modules and then these modules were separated further as shown. The remainder of this section will briefly explain the tests used on the individual modules and the results of these tests.

Hardware Tests. The hardware module was separated into four

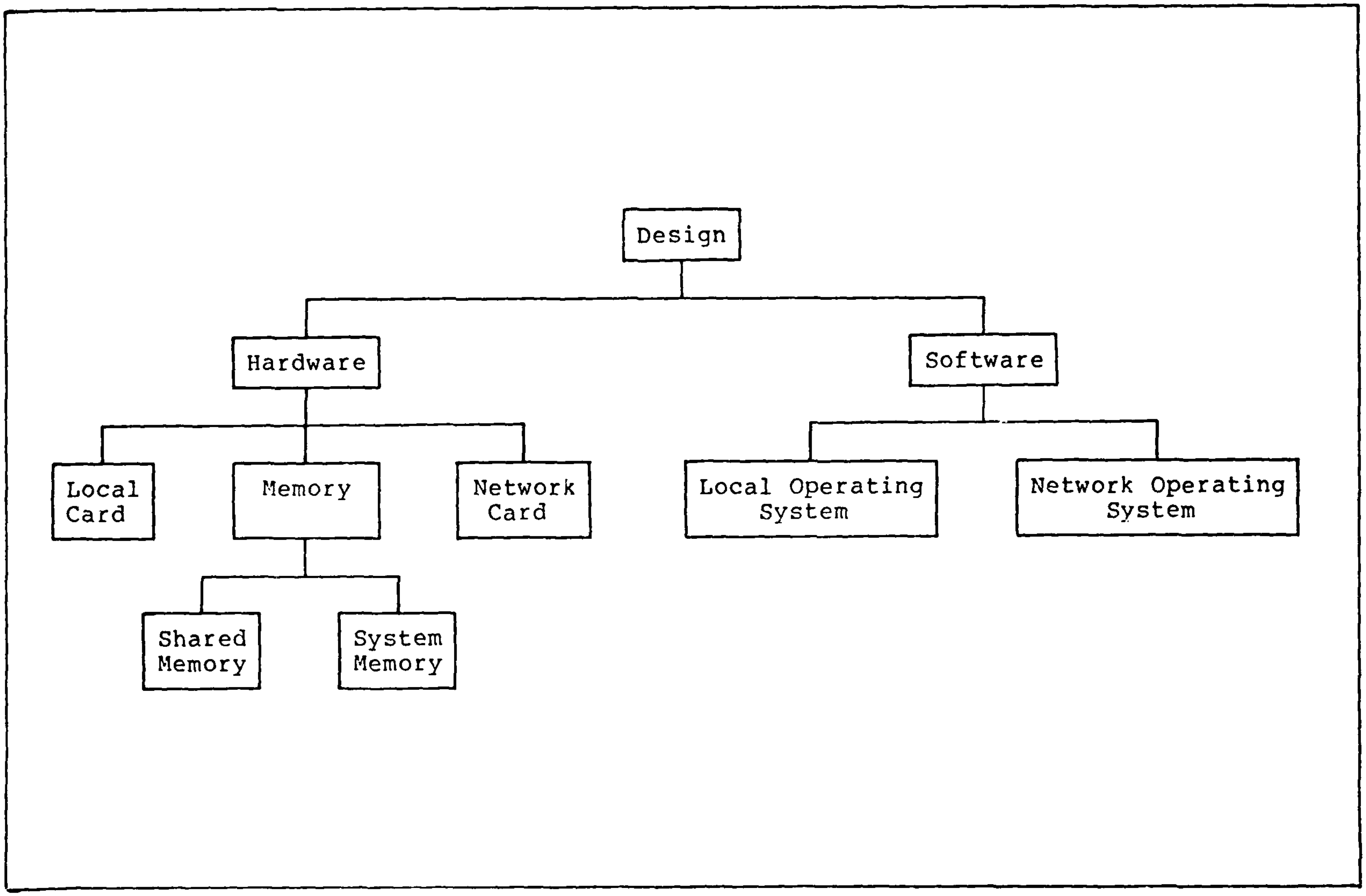


Fig. 8 System Modules

smaller modules the Local Card, the Network Card, the Shared Memory Board and the System Memory Board. The tests for each of these modules will be explained below.

<u>Local</u> <u>Card</u>. The Local Card was separated as shown in Figure 9. The tests on these modules will be performed with simple software routines designed to test each functional

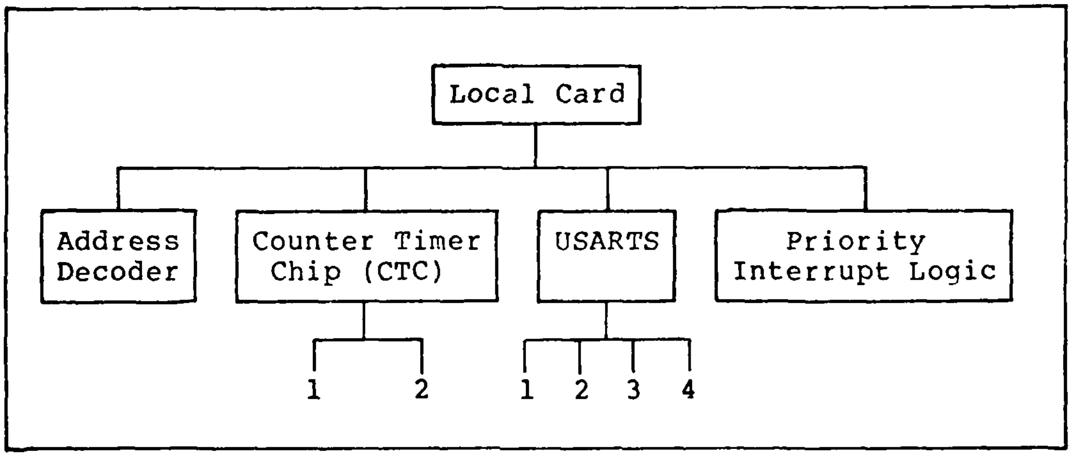


Fig. 9 Local Card Tests

unit shown in Figure 9. These tests were designed for use with an oscilloscope which will be connected to the ouput of the functional unit being tested. Testing will be performed from left to right across the bottom of the figure. This ordering is used because each functional unit on the right may require one or more of the functional units on the left to perform its function. After all of the functional units of the Local Card are tested the Local Card itself is tested with a simple software routine that is designed to demonstrate that information can be passed through the Local Card by means of an interrupt driven routine. The Network Card tests will be explained next.

Network Card. The Network Card functional modules are

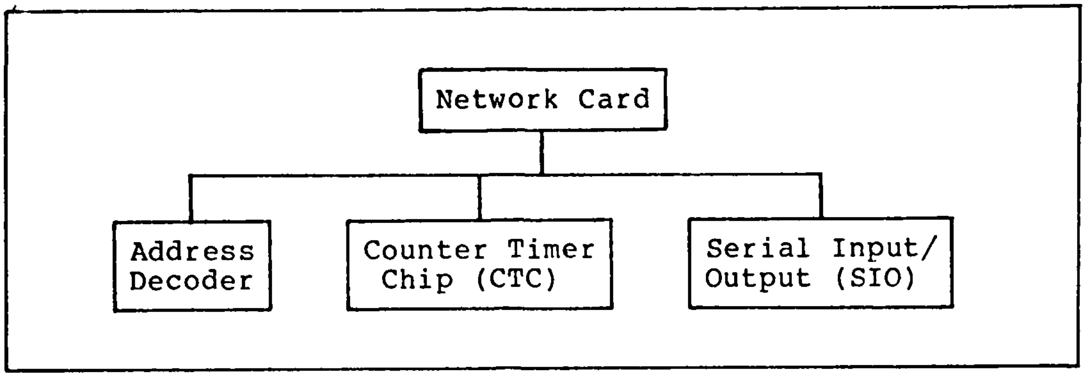


Fig. 10 Network Card Tests

shown in Figure 10. These modules again are shown in order from left to right and are to be tested by simple software routines designed to be used with an oscilloscope. After all of the functional units of the Network Card are tested then the Network Card itself is tested as a functional unit.

Assembly listings of the tests used, for testing both the Local and Network Cards as functional units, are included in Appendix E. The testing of the memory boards will be covered next.

Shared Memory Board. The Shared Memory Board is itself a functional unit and can be tested by demonstrating that both sides of the UNID are capable of reading from and writing to this memory. This test involves writing a known pattern into the memory with one side of the UNID and then determining if this pattern can be read by the other side of the UNID. The test is run from both sides of the UNID to make sure both sides are reading and writing correctly.

System Memory Board. The System Memory Board can be tested with any standard assembly language memory test since the memory on this board is not shared between the two processors. A quick test of this board can be done by using the Fill command in the monitor to fill the memory with some known information and then checking that the information was actually written into the memory.

This concludes the test plan for the hardware of the UNID. The software test plan will be covered in the next section.

Software Tests. The software was separated into two functional units corresponding to the two different operating systems, one for the Local side of the UNID and one for the Network side. These tests can all be performed by loading the entire operating system into the UNID and using the monitor to set the processor registers to the proper values. Then using either the Next (single step) command or setting break points with the Break command, to check for the proper operation of the module. The Local Operating System tests will be explained first.

Local Operating System. The Local Operating System can be separated into three functional modules: initialization, subroutines and the main loop. The subroutine module can be further broken down into the individual subroutines that handle the TTY receive and transmit functions, the memory allocate and deallocate functions and the add to queue and subtract from queue functions. This break down of the Local Operating System is shown in Figure 11.

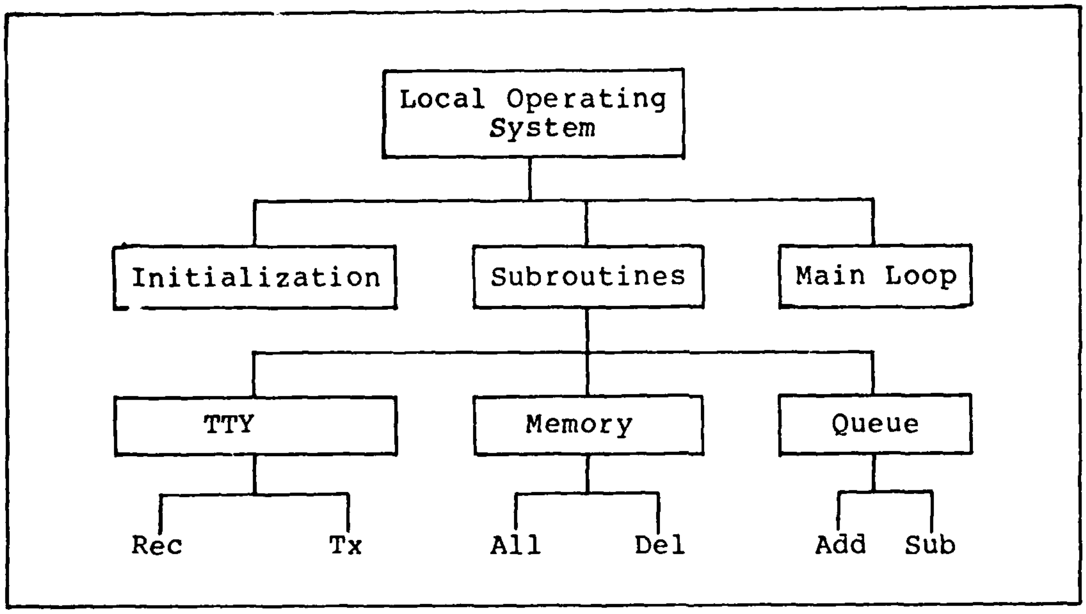


Fig. 11 Local Operating System Tests

For the initialization test each of the individual sections of code used to initialize the processor registers, certain components on the Local Card, and the queues and tables needed for operation, are stepped through to make sure they are performing their desired function. Once these sections of code are operating properly the subroutines of the Local Operating System can be tested.

There are six subroutines that need to be tested for the Local Operating System. These correspond to the six blocks on the bottom row of Figure 11. The TTY transmit and receive subroutines are interrupt driven and can be tested by connecting a terminal up to the Local Card and trying to transmit to and receive from the terminal through the Local Card. The receive function can be tested by performing the initialization and then inputting to the Local Card from the terminal and using the monitor to determine if the receive subroutine is loading the characters into memory properly. The transmit function can be tested by determining if these received characters can be retransmitted to the terminal using the transmit subroutine.

The memory allocation and deallocation subroutines are used to allocate and deallocate a block of memory from the memory table that was built during initialization. These routines can be tested by setting up the processor registers, as they would be when the subroutines are called, then calling the subroutine and using the monitor to check the memory table for the proper operation. The allocation subroutine should remove a memory block address from the table and the deallocation subroutine should put a memory block address back into the table.

The queue addition and subtraction functions are used to add a memory block address to the tail of a queue and to subtract a memory block address from the head of the queue. These functions can also be tested with the monitor by setting up the registers properly and then stepping through the function. The function is checked to be sure it is performing correctly.

After the subroutines have been shown to be operating correctly then the individual sections of the main loop can be stepped through with the monitor. During this single stepping checks will be made to determine if the correct queues and tables are being addressed and if the correct subroutines are being called at the right times.

This will complete the individual module testing of the Local Operating System. The Local Operating System as a whole will be tested in conjunction with the Network Operating System during the full system test. The Network Operating System tests will be explained next.

Network Operating System. The Network Operating System can be separated into three functional modules as shown in

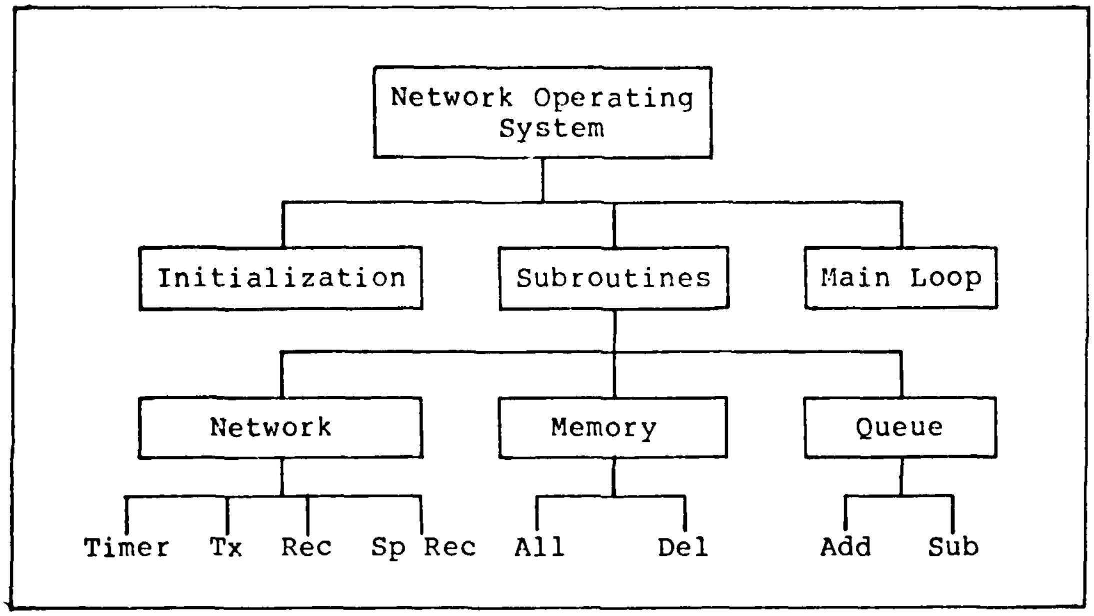


Fig. 12 Network Operating Systems Test

Figure 12. These modules are the initialization, the subroutines and the main loop. The subroutine module can be further separated into the individual subroutines which are the network receive, network transmit, timer, special receive, the memory allocate and deallocate and the queue add and subtract subroutines. The memory allocate and deallocate and the queue add and subtract routines are identical to the routines in the Local Operating System, therefore, their test procedures will not be explained

here.

The test for the initialization section of code is essentially the same as for the Local Operating System. The only changes are that some different processor registers are set, that different queues and tables are set up and the components on the Network Card are initialized. After the initialization testing is complete then the subroutines can be tested.

There are eight subroutines that require testing in the Network Operating System. The tests for four of these has already been explained. The remaining four are the ones shown under the Network block in Figure 12. Of these the receive, timer and the special receive subroutines are interrupt driven. Three of the four subroutines are tested at the same time. These are the receive, transmit and the special receive subroutines. They are tested by connecting the transmitter of the Network Card to the receiver and then running the following test. A block of memory is loaded with known information and the transmit routine is called with the address of this block. The transmit routine should transmit the information while at the same time the interrupt driven receive routine is receiving the information. When the last character is received then the special receive routine should be entered to reset the receiver. The timer subroutine is tested by setting the CTC timer on the Network Card and letting the timer cause an interrupt and then stepping through the timer routine to make sure that it functions properly. Once the subroutines are tested then the main loop can be tested using the same procedure explained for the Local Operating System.

This will complete the individual module testing of the Network Operating System. The whole system can now be tested as a functional unit.

System Test. After the individual modules of the hardware and the software have been tested and are operating properly it becomes an almost trival matter to test the system as a whole. To help see the flow of this testing, some extra test routines have been added to both operating systems that will print on the system console what is happening during system operation. These routines are enclosed between conditional assembly statements in the operating systems source code that can be turned on or off during assembly. It is suggested that these routines be used during any systems' test. To test the system two terminals are connected to the Local Card and the transmitter and receiver on the Network Card are connected together. With this done some information is input to one of the terminals connected to the Local Card. The information input on the first terminal should appear on the second terminal. If this happens then all of the functions of the UNID are operating properly but in a very limited throughput mode. The remainder of this chapter will show the results of the above test plan.

Test Results

This section will briefly describe the results of the tests that were explained in the previous section. These tests were performed throughout the course of this investigation. As hardware tests determined that certain boards or cards were operational then the software for these was tested. This process was continued until all of the tests had been completed.

The remainder of this chapter will cover the results of the tests, not in the order that they were performed, but in the order that they were explained in the previous section.

Hardware Test Results. The results of the hardware tests will be explained below. These tests at times revealed some wiring or design errors; these errors and what was done to correct them will be briefly explained. The results of the Local Card tests will be shown first.

Local Card Tests. The first test performed on the Local Card was the test of the address decoder. The only problem encountered during this test was that no documentation could be found as to what addresses this decoder was suppose to be decoding. Table I shows the port addresses (in hexadecimal) that were found decoded on the Local Card and the devices these addresses are associated with.

Next a test was performed on the Z80 Counter Timer Chip (CTC) (Ref 14). This test showed that CTC 1 was functioning normally. At this time the Local Card has been configured

TABLE I

LOCAL CARD PORT ADDRESSES

Local Card Device	Port Addresses
USART 1	0 - 3
USART 2	4 - 7
USART 3	8 - B
USART 4	C - F
CTC 1	10 - 13
CTC 2	14 - 17
PIC	18

with only one CTC and two USARTs (Universal Synchronous Asynchronous Receiver Transmitters). Therefore, the second CTC could not be tested.

The two USARTs that are configured on the Local Card were tested next. These devices were operating normally but it was discovered that there was no means, on the card, of changing the configuration of these devices. The original design called for the USARTs to be configurable as either Data Communication Equipment (DCE) or as Data Terminal Equipment (DTE) (Ref 5:61-62). To meet this design goal, the outputs of each USART were connected to one side of a wire-wrap socket and the other side of the socket was connected to the Local Card edge connector wire-wrap posts that are connected to the RS-232 lines. This modification will enable the USART to be configured as either DCE or DTE by changing the wiring of a component mount that can be mounted in the wire-wrap socket.

Next the priority interrupt logic of the Local Card was tested. This logic appeared to be functioning normally, but again it was found that part of the original design for this module had not been implemented. This design had specified that the mask word, that is used by the Intel M8214 Priority Interrupt Control Unit (PICU) (Ref 3:6-183 to 6-185), be either hardware or software selectable and neither of these had been implemented (Ref 5:72). Since both implementations required the same amount of rewiring it was decided to implement the software selectable option. This option was selected since it would be easier to change the mask word, if required in the future, by changing software rather than hardware.

With all of the individual modules of the Local Card tested the Card itself was tested. The Card performed it's function properly. The changes that were made on the Local Card are reflected in the schematics of the Local Card which are in Appendix B. The testing of the Network Card will be discussed next.

Network Card Tests. The first module tested on the Network Card was the address decoder. This module tested properly but, again, as on the Local Card no documentation was found on what port addresses the decoder was decoding. Table II shows these port addresses (in hexadecimal) and the Network Card devices associated with these addresses.

The Z80-CTC on the Network Card was tested next. This device was operating properly as a clock input to the

TABLE II

NETWORK CARD PORT ADDRESSES

Network Card Device	Port Addresses
SIO	0 - 3
CTC	4 - 7

Z80-SIO (Serial Input/Output) (Ref 10) but was not operating correctly in the interrupt mode. The problem found was that the interrupt line going to the processor had not been connected to either the Z80-CTC or the Z80-SIO which also required the connection. After this signal's line was connected the Z80-CTC operated properly.

Next the Z80-SIO was tested. This chip was operating properly in the asynchronous mode but was not resetting after an interrupt in the interrupt driven synchronous mode. The first problem found was that the Z80-SIO itself was defective. After replacing the Z80-SIO there was still a problem during interrupts in that the first character was the only one that would cause an interrupt when the Z80-SIO had been programmed to interrupt on every character received. It was found that the Z80-SIO was still not resetting itself after an interrupt as it was suppose to, therefore, it never could generate any interrupts after the first. The Z80-SIO is designed to reset itself when it decodes a RETI (RETurn from Interrupt) instruction on the data bus (Ref 10:4). The problem being encountered was caused because the Z80-SIO never decoded the RETI instruction because the buffers for the data bus on the

Network Card were disabled. These buffers as designed were disabled except when a port address on the board was decoded. Since this would not allow the Z80-SIO to monitor the data bus, the circuit used to enable and disable the data buffers was redesigned. The new design reduced the gates used for this circuit from 15 to 3. This reduction was made possible by taking full advantage of the fact that the Z80-SIO was designed to operate with the Z80 processor. The Z80-SIO was tested with the redesigned circuit and found to be functioning properly in all modes.

With all of the individual modules of the Network Card tested the Card itself was tested. The Card performed it's function properly. The changes made to the Network Card are reflected in the schematics in Appendix B. The tests of the two memory boards will be discussed next.

Memory Boards. The Shared Memory Board was tested with 5K bytes of memory and found to be operating properly. As more memory is added to this board it will have to be tested again. The System Memory Board has not been implemented at this time, therefore, no tests were performed on it. The testing of the system software will be discussed in the following section.

Software Test Results. The results of the tests of the system software will be covered in this section. In some cases these tests have not been completed or have not been started; this will be indicated where this is the case. The Local Operating Systems tests will be covered first.

Local Operating Systems Tests. The initialization module was the first tested in the Local Operating System. The trouble encountered during this test was that there was no way of single stepping through the code because the code changed the value in the I register (Interrupt register). This caused the Next command in the monitor to function improperly because it used interrupts. This problem was solved by adding a short piece of code, that is for test purposes only, in the beginning of the Local Operating System. This code loads the correct values into the Local Operating System's interrupt vector table to cause the Next command to function properly. After this was done the initialization module functioned properly.

The memory allocate and deallocate subroutines were checked next. These routines functioned as they were suppose to. The queue add and subtract functions were checked and they also functioned properly.

Next the TTY receive subroutine was tested. After a couple of minor changes in the code this routine was functioning properly. The TTY transmit subroutine has not been tested at this time.

Only the first couple of sections of the main loop of the program have been tested at this time. The main loop is functioning properly as far as it has been checked. The testing of the Network Operating System will be discussed next.

Network Operating System Tests. The initialization of the Network Operating System was tested first. This required that the same section of code be added as was added to the Local Operating System to make the monitor Next command function. The initialization coding for the Z80-SIO was also changed to reduce the amount of code needed for this function and also to make the code more understandable. After these changes the testing indicated that the initialization module was operating properly.

The memory allocate and deallocate subroutines were tested and functioning properly. The queue add and subtract routines also functioned properly when tested.

The network subroutines were tested and they all performed properly after minor code changes. The main difficulty encountered with the network subroutines was learning how to program the Z80-SIO properly. The Z80-SIO is a very complicated IC and has many operating modes. The documentation supplied by Zilog for this chip is not at all clear in some places and doesn't go into enough detail to properly program the chip without a lot of trial and error. A source of information that was found very useful when programing the Z80-SIO was the explanation of the chip given in An Introduction to Microcomputers Volume 3 (Ref 3).

The main loop of the Network Operating System was now tested and appears to be operating properly with the limited testing that could be done without a run of the full system.

This concludes the software tests that have been accomplished on the system at this time. The next section contains a summary of what has been discussed in this chapter.

Testing Summary

This chapter has explained the test plan that was used for the testing of the UNID and also the results of the tests that were performed. Also presented was a short explanation of the design, wiring and coding errors found and the manner in which these errors were corrected. The testing of the hardware has shown that it is now operational. The software is close to being operational but still requires some additional testing of the Local Operating System. Once the software has been completely tested the entire system should be ready for testing.

Appendix E contains assembly listing of the test routines used to test the Local and Network Cards as functional units.

VI. Summary and Recommendations

The primary objective of this thesis was to have an operational Universal Network Interface Device. This involved the modification of the existing monitor, the development of two operating systems, the development of certain new hardware, the redesign of some existing hardware and the testing of both the hardware and software. The remainder of this chapter contains a brief summary of the results obtained and some recommendations for follow-on efforts.

Thesis Results

This thesis has resulted in an operational and tested device as far as hardware is concerned and two partially tested software operating systems. This section will give a brief overview of these results.

The monitor that was upgraded during this thesis has permitted the testing of the hardware and software of the UNID in a very efficient manner. Of the two new memory boards that were designed, the Shared Memory Board has been implemented and tested and the System Memory Board has been partially implemented with no testing. The previously implemented boards have all been tested and shown to be fully operational. The two test operating systems that were developed, though not sufficient for an operational UNID, will provide a good framework for operational systems. The Network Operating System has been tested up to the point of full systems testing and the Local Operating System has been tested with the exception of one subroutine and a few sections of code in the main loop of the program. Once the Local Operating System tests have been completed the UNID will be ready for a full systems test.

Recommendations

The recommendations for the Universal Network Interface Device involve the completion of testing for the device, the implementation and testing of the device in an operational network and some modifications of the monitor.

Completion of the testing of the UNID will involve the full systems tests, as described in the chapter on testing, after the Local Operating System tests have been completed. This testing should be a relatively short project in itself, but will require that the person performing the tests have a complete understanding of the device.

Implementation of one or more UNIDs in an operational network will involve the development of two operating systems for each UNID used and the possible construction of one or more additional UNIDs. An investigation into this would require a full understanding of the UNID plus research into different network protocols that could be implemented in the network.

Modifications to the monitor would involve the upgrading of some the commands to make them easier to use and the addition of some convenience features. For example, the Next command needs to be modified so that it will not require the use of interrupts for its operation and a backspace feature would be a helpful modification to some of the existing commands.

It is hoped that this thesis has taken the UNID to the point where future investigations will be able to concentrate more on the software to make the device operational than on the hardware.

Bibliography

1. 1842 EEG/EEIC TR 78-5. An Engineering Assessment Toward Economic, Feasible and [illegible] Base-Level Communications through the 1980's. [illegible] AFB, Missouri: 1842 Electrical Engineering [illegible], [illegible] OCT 1977.

2. Brown, Eric F. "Prototype [illegible] Interface Device", Master's Thesis. [illegible], [illegible]: Air Force Institue of Technology, [illegible].

3. Kane, Jerry. An Introduction [illegible] Volume 3 Some Real Support [illegible]. [illegible]: Adam Osborne and Associates, Inc., [illegible].

4. Loewer, Bob. "The Z-80 In Parallel," Byte, [illegible]:60-63, 174-176 (July 1978).

5. Sluzevich, Sam C. "Preliminary Design of a Universal Network Interface Device", Master's Thesis. Wright-Patterson AFB, Ohio: Air Force Institue of Technology, December 1978.

6. Texas Instruments, Inc. The Line Driver and Line Receiver Data Book. Dallas, Texas Instruments, Inc., 1977.

7. Texas Instruments, Inc. The TTL Data Book (Second Edition). Dallas, Texas Instruments, Inc., 1976.

8. Zilog, Inc. Report on the Programming Language PLZ/SYS. Cupertino, California: Zilog, Inc., June 1978.

9. Zilog, Inc. PLZ Version 3 User's Guide. Cupertino, California: Zilog, Inc., July 1979.

10. Zilog, Inc. Product Specification Z80-SIO. Cupertino, California: Zilog, Inc., March 1978.

11. Zilog, Inc. The Z80 Family Program Interrupt Structure. Cupertino, California: Zilog, Inc., October 1977.

12. Zilog, Inc. Z80 Assembly Language Programming Manual. Cupertino, California: Zilog, Inc., January 1978.

13. Zilog, Inc. Z80-CPU/Z80A-CPU Technical Manual. Cupertino, California: Zilog, Inc., 1977.

14. Zilog, Inc. Z80-CTC/Z80A-CTC Technical Manual. Cupertino, California: Zilog, Inc., 1977.

15. Zilog, Inc. Z80-MCB Hardware User's Manual. Cupertino, California: Zilog, Inc., January 1978.

16. Zilog, Inc. Z-80 MCB Software User's Manual. Cupertino, California: Zilog, Inc., May 1978.

Vita

Lee R. Baker was born on 24 January 1951 at Schenectady, New York. He graduated from Chestertown High School in 1969. He attended Michigan Technological University from which he received a Bachelor of Science degree in Electrical Engineering in June 1973. Upon graduation he received a commission in the USAF and went to Undergraduate Pilot Training at Vance AFB, Oklahoma. Following graduation from pilot training he was assigned to Pope AFB, North Carolina flying C-130s from February 1975 to May 1979. He entered the Air Force Institute of Technology in June 1979.

Permanent address: P.O. Box 205

Pottersville, N.Y. 12860

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AFIT/GCS/EE/80D-4	2. GOVT ACCESSION NO. AD-A100	3. RECIPIENT'S CATALOG NUMBER 287
4. TITLE (and Subtitle) Prototype and Software Development for Universal Network Interface Device		5. TYPE OF REPORT & PERIOD COVERED MS Thesis
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) LEE R. BAKER Capt USAF		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Air Force Institute of Technology (AFIT/EN) Wright-Patterson AFB, Ohio 45433		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS		12. REPORT DATE December 1980
		13. NUMBER OF PAGES 70
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

Approved for public release; IAW AFR 190-17 Fredric C. Lynch 11 JUN 1981

FREDRIC C. LYNCH, Major, USAF
Director of Public Affairs

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Data Processing
Message Processor
Networks
Protocols

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This thesis describes the third phase of the development and testing for a flexible message processor designed for computer communications network applications. This message processor is microcomputer based and is called a Universal Network Interface Device. The thesis describes the upgrading of the two system monitors, the design of two

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

memory boards, the development of a test plan for testing the device, and the results of the testing that was performed. The device employs two microcomputer boards, a local card, a network card and two memory boards to provide the capability to interface local users to a computer communications network. The device also provides the capability to act as the node connecting two networks operating under dissimilar protocols.

DATE
FILMED

– 8