DESIGN AND ANALYSIS OF A MULTI-MEDIA, MULTI-ACCESS DATA NETWORK

by

John Christian vom Lehn

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREES OF
BACHELOR OF SCIENCE
and
MASTER OF SCIENCE
at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May, 1983

Signature of Author . . . . . .
Department of Electrical Engineering and Computer Science
May 1, 1983

Certified by . . . . . .
Professor Jeffrey H. Shapiro, Academic Thesis Supervisor

Certified by . . . . . .
C.M. Puckette, GE Corporate Research and Development Center

Accepted by . . . . . .
Professor Arthur C. Smith, Chairman,
Departmental Committee on Graduate Students

DESIGN AND ANALYSIS OF

A MULTI-MEDIA, MULTI-ACCESS DATA NETWORK

by

John Christian vom Lehn

Submitted to the Department of Electrical Engineering and Computer Science on May 3, 1983 in partial fulfillment of the requirements for the degrees of Bachelor of Science and Master of Science in Electrical Engineering.

ABSTRACT

A local area network was designed, constructed, analyzed and improved. The network employed a modified token passing protocol, similar to a round robin approach, implemented in a bus structure. The network was designed to use either fiber optics or wire pair, or a combination of both, as the network transmission medium. Network stations were microprocessor-based, consisting of a central processor, two slave processors and a shared memory resource.

After an original working model was successfully constructed, the performance of the network, notably its efficiency, access delay, and throughput were carefully analyzed. Certain aspects of the protocol and hardware design were found to cause significant downgrading of performance. Based on the analysis, a revision of the network was proposed. As an empirical measure of the network's performance, a data traffic simulator was constructed and used to verify the conclusions of the analysis.

Work on the project was conducted at General Electric's Corporate Research and Development Center in Schenectady, New York.

Thesis Supervisors:

Dr. Jeffrey H. Shapiro
Associate Professor of Electrical Engineering

Mr. C.M. Puckette
General Electric Company
Corporate Research and Development Center

Table of Contents

List of Tables

List of Illustrations

Acknowledgements

Grateful acknowledgement is made of the contributions of Dr. Sanjay K. Bose, who designed the basic protocol for the network and supported the initial software programming, and to Mr. Eugene J. Orlowski, whose continuing technical assistance through the construction and demonstration of the network has been invaluable. Acknowledgement is also made of the assistance of Mr. C.M. Puckette of General Electric and Professor Jeffrey Shapiro of MIT whose patient supervision of this project provided a constant positive incentive.

I. Background and Terminology

A. Introduction

The simplest function of a data communications network is to transport data from one point to another. As data sources have become more complex, however, data communication networks have been required to provide a wider range of functions, thus necessitating a higher degree of intelligence in the network itself. The concept of an intelligent network is the foundation of the emerging industry of so-called "local area networks".

To date, there has been a great deal of work done in developing the concept, but little has been accomplished in the area of standardization [1]. Every major manufacturer in the computer industry, hoping to seize a large share of the new market, has proposed at least one network and cross-compatibility has been given little attention [2]. In addition to the competition roadblock, standardization has also been hampered by the fact that a major portion of these networks is not hardware but software, thus allowing a much wider range of possibilities.

Because of the relative newness of local area networks and the absence of any standard design, the creation of a local area network presents an interesting free-form problem. The network may be specified from the protocol level to the actual hardware based on design objectives alone; the problem becomes as much to determine what is needed as what can be done. Such a project was initiated at General Electric's Corporate Research and Development Center in 1981. This thesis report describes the creation of a local area network, from the initial specifications, through prototype construction, to performance analysis and construction of an improved network.

This report contains some limited background informa-

tion as well as a record of the author's actual research. The remainder of this section is devoted to establishing needed terminology for description of the network. Section II discusses the general goals and limitations for local area network design and the reasons for choosing various system attributes in the prototype design. Sections III and IV are devoted to a general description of the prototype network, from a system and hardware standpoint, respectively. Sections V and VI detail the theoretical and actual performance of the prototype network, and Section VII describes changes made to the original network based on the analysis made. Section VIII is devoted to summary, including the applicability and future possibilities for the network. Two appendices are also included, the first describing the approach used in the software programming of the network, including example programs, and the second detailing the implementation of the network using both wire pair and fiber optics as the network medium.

The author was involved with the project from its beginning in June 1981, during which time the design objectives described in Sections II and III were determined, in concert with Dr. Sanjay Bose and Mr. Eugene Orlowski of General Electric. The author completed the hardware design is described in Section IV in 1981, and returned to the project in May 1982 to conduct the research described in the remaining sections of this report.

B. Descriptive Terminology

While one of the goals of this report is to avoid useless jargon and buzzwords, some uniquely-defined terminology will aid in describing the project. This terminology will be explained in the context of the basic function and structure of a local area network. First, and perhaps most difficult, is the definition of the term

"local area network" itself. Several papers have been written to this end [3], hoping to establish a basis for classification, but for the purposes of this report, a strictly functional definition will suffice: a local area network is system used to support data communication among several users, using a shared communication medium. Users here may include any of a wide variety of data sources with varying degrees of intelligence, or decision-making capabilty, including computers, data terminals, computer peripherals, instrumentation, and numerically-controlled machinery. The communication medium may be free space, single or multi-conductor wires, cables, or waveguides.

Data in the communication medium may be broadcast to all units of the network, or forwarded from one unit to another. Broadcast type networks include bus structures and dendritic (tree-like, with branches) structures [4]. Forwarded-data networks include net structures, where data can take any of several paths to a given destination, and daisy-chain or ring structures [5].

The characteristic function of a local area network is found in the interface between the user and the shared communication medium. A common goal for a local area network is "transparency", meaning that the user need not be concerned with the network's operation (the network is unseen, or transparent); the user merely gives the network data with the understanding that it will be sent to the appropriate destination without the need for any additional instructions. All local area networks will provide intelligence for arbitration among users as they require use of the shared medium, as well as some mix of functions for the interface to the user. As shown in Figure I.1,these interfacing functions may include buffering (temporary storage of data), packetizing (division of data into standard length units), error-checking (inclusion of a

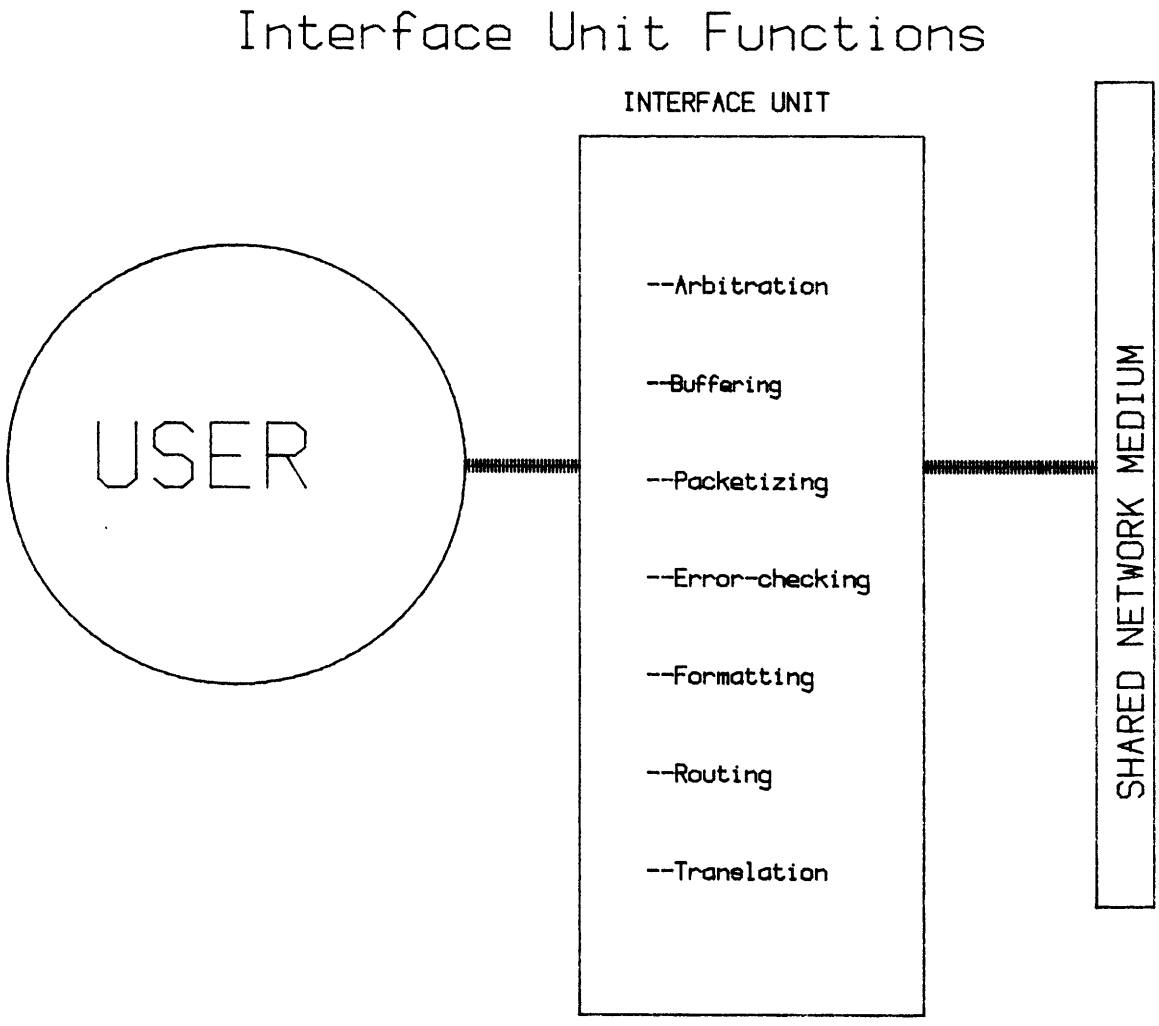


Figure I. 1: Interface Unit Functions

calculation based on the input which can be re-calculated at the destination), formatting (conversion of data into a standard form), routing (choice of path to destination), and translation (one code or data format to another). Obviously, for the network to achieve transparency, the network must perform a set of functions on data received from a source, or sender, and then invert these functions before data is relayed to the destination unit.

As has already been implied, there must be some set of rules that govern access to the shared medium. These rules are called the network protocol. In general, the network protocol can be viewed in a layered approach, covering everything from actual connections to the medium to commands given to the network for given applications [6]; however, for the purpose of this report, network protocol refers only to the rules governing access at the system level.

There are as many different network protocols as there are networks, yet there are two basic types worth distinguishing: random access and deterministic. A random access protocol allows units to send information as soon as they receive it, subject to some restrictions. Such a protocol is Carrier-Sense Multiple Access (CSMA) [7], in which a station with data to send first checks to see if the shared medium is being used, and if it is not, the station sends the data. Different propagation times among the various units may result in two messages being sent at once, causing a "collision." The protocol has further stipulation regarding operation in case of a collision, determining which unit may send again and which must wait. A deterministic protocol has rules which never allow two units to transmit at once in correct operation. Examples of such protocols are Master/Slave protocols, in which a central unit, the master, commands other units, the slaves,

to send or receive data. Other examples are token-passing protocols, where the right to send data is contained in possession of the "token," which is a signal or code held and then passed among the units in the network [8].

Finally, some terminology is necessary for the description of the network units. The basic unit of the network is the interface unit. This unit contains all necessary intelligence for sending data and receiving data from the user, and in many cases, contains all intelligence necessary for the operation of the network. Other networks may include control units which ensure correct operation of the network. These units are generally much fewer in number than the interface units and act in a monitoring and command fashion.

This, then, is the composition of a local area network: interface units, optional control units, and a shared medium. The interface units connect users to the shared medium, allowing communication with other units. Access to the medium is governed by the protocol, which is implemented by the interface units and the control units.

Section II. Considerations for Network Design

This section examines the design objectives for the prototype network and the reasons for many of the design choices. It should be pointed out that a considerable amount of the fundamental decisions described are subjective; other designers might attack the same problem in an entirely different way. The subjectivity arises out of the fact that, as with all engineering problems, the solution lies not in discovering the single perfect design, but in striking a balance between conflicting performance goals. Thus, this section is largely an account of the trade-offs associated with local area network design.

The most obvious design trade-off involves simplicity: the simpler the network, the lower its cost, the smaller its circuit size, and in general, the greater its reliability. There is little need, however, to extrapolate the pure simplicity argument very far; a network must provide at least basic functions in order to be superior to conventional "dumb" circuitry. In general, these advantages are had by high speed and efficient use of network capabilities. More elegant designs can be justified by other features, such as interface flexibility, failure protection, easy expandability, and varying degrees of network transparency. An initial decision, therefore, is the scope of the network: what degree of elegance is justified by the projected market? For the prototype network, the approach was slanted toward simplicity, based on a desire to produce a working model as soon as possible, and also based on an assumption that many potential customers would be willing to forego advanced features in favor of lower price. Thus the goal was set to achieve the best results from a small-scale system.

With the simplicity trade-off settled, it remained to

make some more quantitative decisions. Since the projected market was not the mainframe-to-mainframe network market which requires more elegant features, but the smaller scale control and instrumentation market, inquiries were made in regard to data requirements. A study [9] showed a need for regular transmission of short data messages, as opposed to infrequent, long messages. The large majority of messages were small (less than 256 bytes) and most devices were incapable of assimilating data at rates faster than fast terminals (about 20 kilobytes per second). A wide variety of user types was projected, with most receiving much more data than they send. Based on these reports, the design became more specified: a medium-speed, short message, medium utilization network.

The projected market also specified the operating environment: typically a harsh factory environment, where conductors might occasionally be severed, and where electro-magnetic interference is often a problem. Reliability needed special consideration. Finally, price was also an important issue: it was necessary to keep the network's cost well below that of the more elegant networks, which typically cost in the $800 to $1000 range per unit [10]. Clearly, a small price differential would not justify the reduced performance; the network needed to be about an order of magnitude less expensive.

At this juncture, some actual specification could be made. Because of the simplicity and price requirements, a medium-level (about 100K bits per second) network data rate was favored. Because of the relatively constant level of utilization and the acceptability of the lower network rate, a deterministic protocol was preferred. This is because, as will be shown in Section V, a deterministic protocol is capable of higher efficiency than random access protocols which exhibit instability beyond a certain level

of utilization. Although arguably more complex to implement, deterministic protocols offer the promise of more efficient use of slower transfer rates. The projected environment indicated a need for noise-immune fiber optics in some if not all parts of the network, and suggested advantages to a bus or dendritic structure over a ring structure, since a ring network would be rendered non-functional from a single break.

Further study indicated that a master/slave protocol was not preferred because of a perceived need for many units to communicate directly with one another; the necessary intervention by a master unit would cause a serious loss of efficiency. On the other hand, the need for reliable operation warranted some kind of monitoring function, supplied by a single control unit. The high incidence of "dumb" users indicated a need for a concentration of intelligence in the user interface to allow for a flexible exchange of data. The number of users was seen to vary widely, but many instances required only a small number of users, thus leading to the four stations of the prototype network. The combination of all these factors led to the initial specification shown in Table II.1; the actual implementation listed is the product of the final specifications described in sections III and IV.

Table II.1 Specifications of the Prototype Network

	Initial	Actual
Network Structure	Bus	Bus
Network Protocol	Deterministic	Token-Passing
	Peer-to-peer	Broadcast mode
	Central control	Single control unit
Network Medium	Fiber optics	Fiber optics
Network Interface	Flexible	Dedicated processor
Message format	--	SDLC frames
Acknowledgement	--	Positive only
Number of BIU's	< 32	4
Network Data Rate	100K bits/sec	64K bits/sec
Interface Data Rate	9.6K bits/sec	19.2K bits/sec
Interface Functions		
Packetizing	Var. length	SDLC format
Error-checking	Shift and add	16-bit FCS
Translation	ASCII encoding	ASCII only
Alternate routing	none	none
Max message length	256 bytes	256 bytes
Buffer size	--	1024 bytes
Interstation distance	100m	100m

III. Network Protocol

A. Bus Interface Units

As was stated in Section II, the desired protocol was one which would be deterministic, support peer-to-peer communication (as opposed to a master/slave arrangement), and operate in a bus structure. Although the first two requirements would seem to suggest a token-passing protocol, the third does not: a token-passing protocol is typically implemented in a ring structure. In that typical implementation, the token is actually no more than a message header. When an individual unit detects this unique header, it knows that it may append whatever messages it may have to send. Data flows in only one direction along the ring, and when a station sees a message addressed to it, it takes the message out of circulation by storing it and not forwarding it as it passes the collection of messages and the token to the next station. Since the inferiority of the ring for this design was sufficiently demonstrated, some changes were necessary in the token-passing scheme.

The principal difference between the ring and bus structures is access to data. In a ring, a station only receives from the preceding station; thus "possession" of the token is a logical concept. In a bus structure, all data is broadcast, that is, all stations receive the same data at roughly the same time. Thus, possession is not a viable concept for the bus structure. The solution to the problem lies in a redefinition of the right to send data. Fundamentally, all that is required is that each station have some unique condition, which it and all stations on the network will recognize, that entitles it to send data. The simplest form of this is a round-robin approach, each station transmitting in a pre-defined order. All that is

then necessary is some way for each station to keep track of what the order is, when it will have the opportunity to transmit, and when a particular station has completed activity in its present turn. Such a procedure is the basis for the algorithm used in the prototype network protocol.

Each bus interface unit (hereafter BIU) is assigned a unique address, called MYADD, in the range 0 to 255. Each unit also keeps a count, called COUNT, of the number of turns completed. When COUNT matches MYADD, the BIU knows that it has exclusive right to transmit. It transmits whatever messages it may have received from the user, waits for acknowledgement from the various destinations, and then sends the token message, which indicates completion of the turn. If the BIU has no messages to send, it merely sends the token message. Each BIU also keeps a value called LIMIT, which is the largest possible value for COUNT. When COUNT exceeds LIMIT, the BIU resets COUNT to zero, and the cycle of turns begins again. The operation of this protocol is illustrated in Figures III.1-5.

The protocol as stated to this point deals only with access to the network medium. Of course, while the BIU is interacting with the network medium, it must at the same time be interacting with the user. This requirement will be more fully examined in Section IV, but it can also be seen in the flowcharts. In Figure III.1, it can be seen that the BIU must maintain a constant attention to the network medium, checking for an input. When an input is received, the type of data received is determined and appropriate action taken. In the case of a message received, the network sends an acknowledgement to the sender, and then must wait until the user is not busy before forwarding the message. For instance, the user may be in the process of transferring data when a message is received. Other network

BIU Protocol

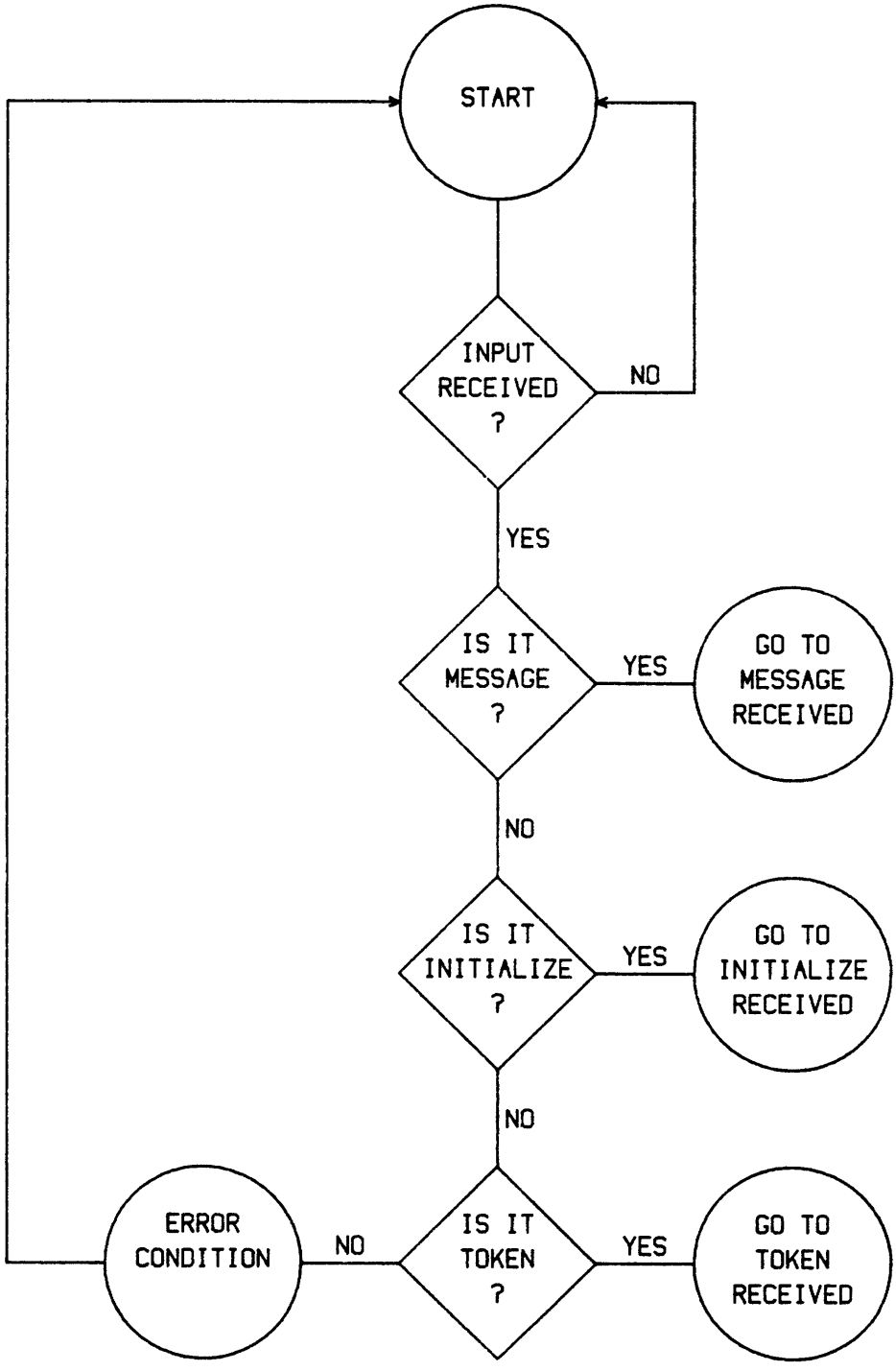


Figure III.1: BIU Protocol

BIU: Message Received

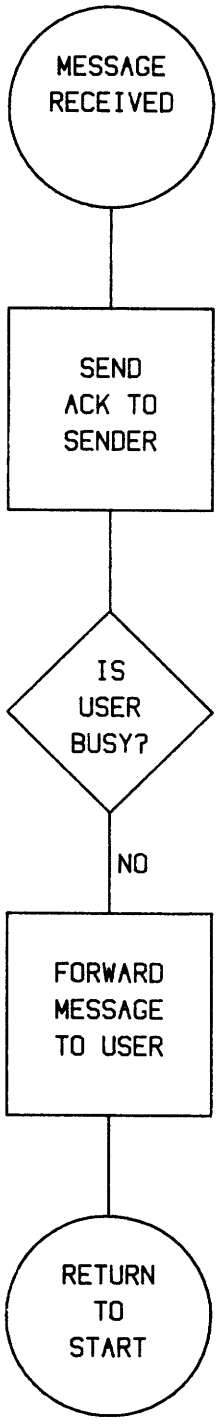


Figure III.2: BIU Protocol--continued

BIU: Initialize Received

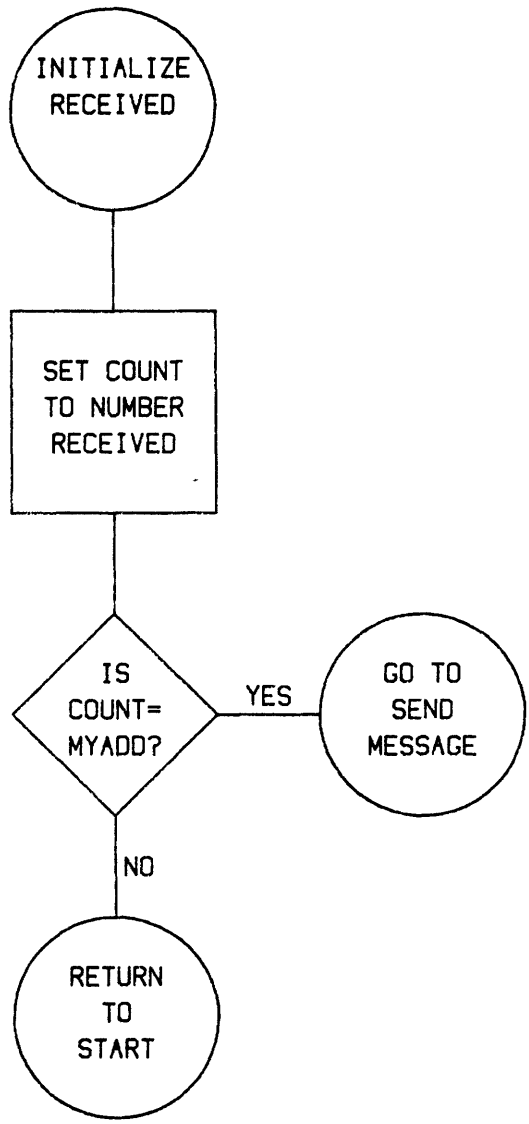


Figure III.3: BIU Protocol--continued

BIU: Token Received

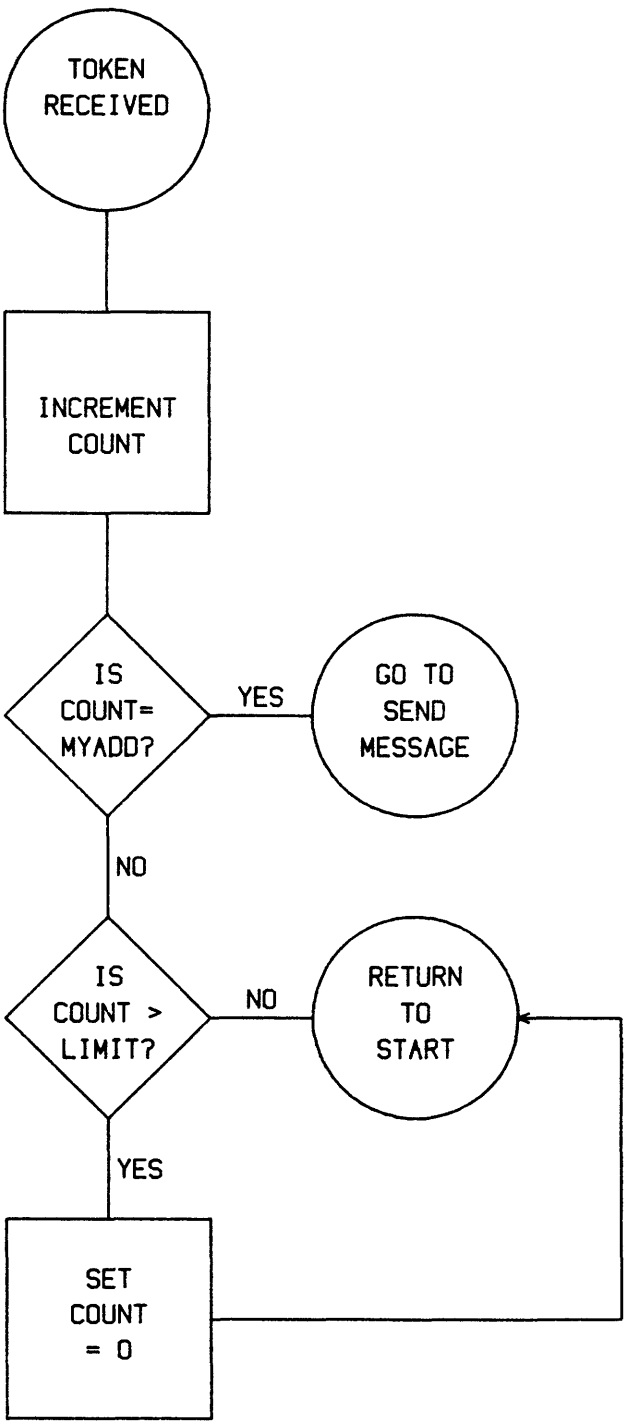


Figure III.4: BIU Protocol--continued

BIU: Send Message

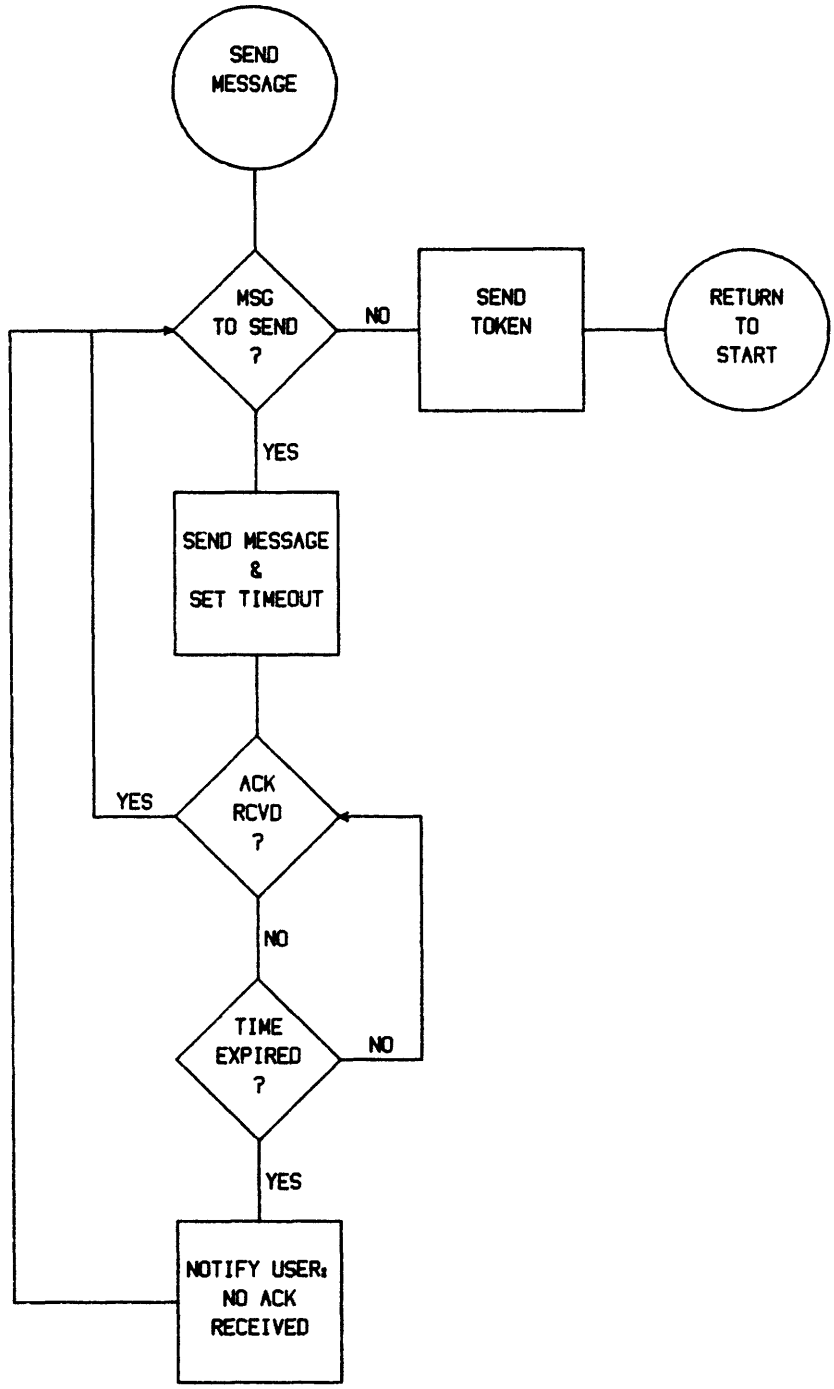


Figure III.5: BIU Protocol concluded

CCU Protocol

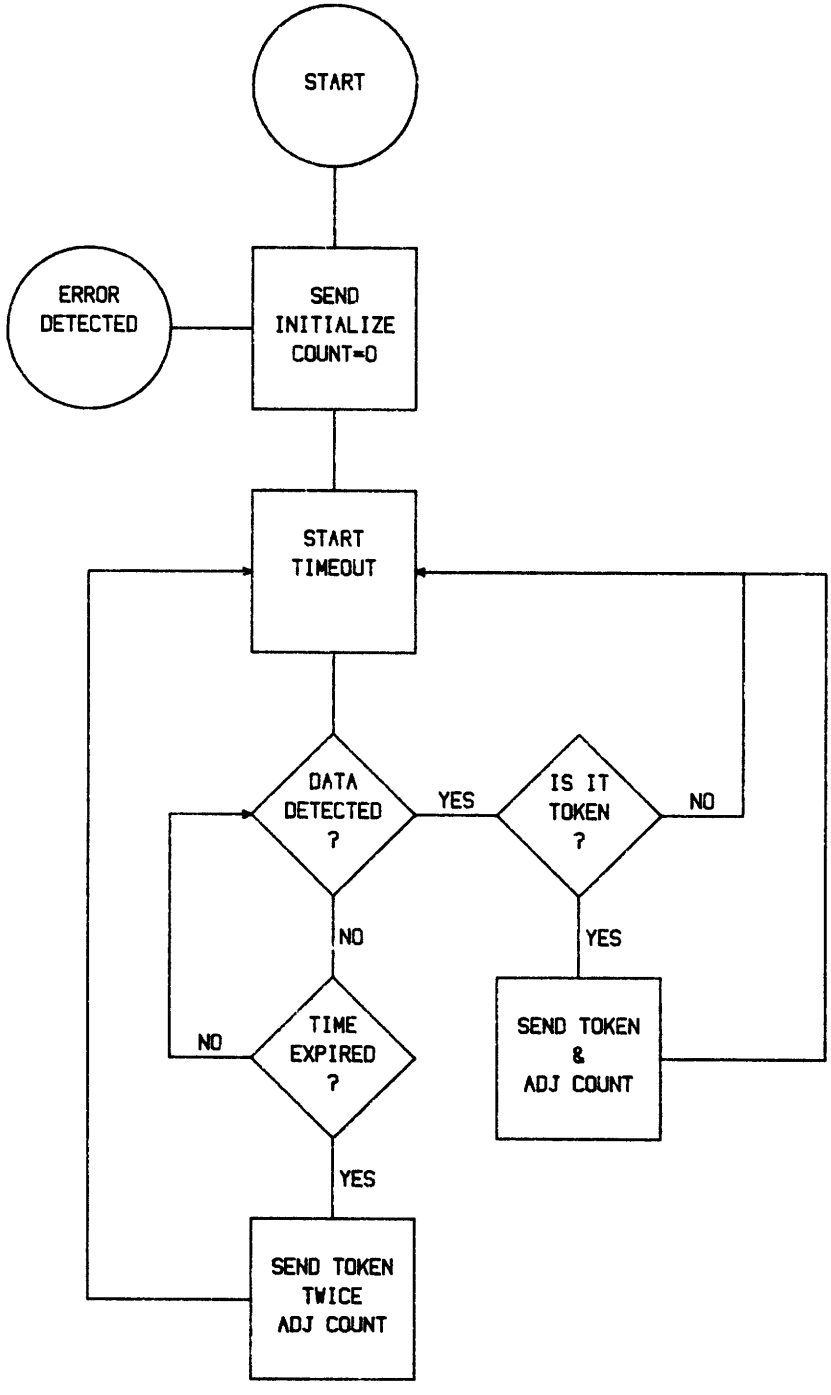


Figure III.6: CCU Protocol

activity remains transparent to the user, except in the case when the post-transmission timeout expires before an acknowledgement is received (see Figure III.5). In that case, the user is notified; otherwise, it may be assumed that transmission of all prior messages was successful.

The inclusion of the central control unit (hereafter CCU) provides the opportunity for more efficient operation as well as greater reliability. Because of a perceived need for quick intervention in the case of error (for instance, two BIU's transmitting at once because of a mistaken calculation of COUNT), the CCU is given a high level of involvement. As seen in Figure III.6, the CCU contains a constant monitoring function and intervenes after every transmission of the token. Whenever the CCU detects the error condition of two active transmitters, it re-initializes the network. The CCU always is in control when COUNT is odd, meaning correct operation of the network is never more than one token away. After each BIU completes a transmission turn, the CCU has the opportunity to take any necessary correcting steps. In normal operation, the CCU sends a token immediately following reception of a token from a BIU. In the event of a BIU failure, the CCU's timeout will expire without detection of any transmission. In this case, the CCU merely sends a token in lieu of the failed unit plus a second token to advance COUNT to the next BIU address, and normal operation continues without a re-initialization of the system. For fastest and most reliable operation, the CCU was not allowed to support a user in the prototype network; however, there is no reason why the CCU's function could not be performed by an interface unit.

The high level of involvement of the CCU also leaves the opportunity for more complex interaction in scheduling of turns. Although the network must be "fair", i.e. no BIU

may be permanently locked out from transmitting, a particular application may require certain users receiving higher priorities or more frequent turns. This may be implemented by use of the CCU's initialization command, sent during the time when COUNT is odd. Instead of allowing COUNT to increment to the next BIU's address, the CCU may send the initialization message, resetting the value of COUNT to the address of the station requiring higher priority. Further, the CCU may be used to keep track of failed units, automatically skipping their turn the majority of the time and thus avoiding the longer delay imposed by waiting for a timeout. Finally, a particular BIU might request a temporary increase of priority by communication with the CCU in a reservation-type procedure. While none of these features was implemented in the prototype network, the CCU involvement was designed with them in mind for future development.

Section IV. Hardware Implementation

The hardware implementation of the network can be broken down into three tasks: determination of necessary functions, matching function and actual integrated circuits, or chips, and specifying interconnection among all chips. As has been noted previously, priority was given to creating a working model as fast as reasonably possible. In the hardware design, however, this goal was balanced against a desire to allow for a large amount of enhancement without a hardware overhaul. As a result, the minimization of chip count has been sacrificed in favor of anticipated improvements. This section describes the transition from protocol to actual circuit. The specifics of the receivers and transmitters used to implement the shared medium are not covered in this section; Appendix B contains a description of the implementation of the bus in both fiber optics and wire pair. For the purposes of this section, bus transceivers are assumed to exist, requiring only synchronous data (NRZ data and a clock) and simple control lines (Ready to Send, Clear to Send, and Carrier Detect).

A. Determination of Necessary Functions

The determination of necessary functions does not constitute an actual flowchart, but a rough estimate of the capabilities needed and an indication of the complexity involved. This estimate, as seen in Table IV.1, can be viewed from the reception and transmission of data on the shared medium to the exchange of data with the user.

Starting at the most elementary level, some part of the hardware must be able to transmit and receive synchronous data and perform the necessary manipulation of bus transceiver control lines. Since data on the bus arrives

Table IV.1 Necessary Functions for Network Units

Bus transceiver management
  Data input/output
  Transceiver clock
  Control line management

SDLC formatting
  Flag appending
  Bit stuffing
  CRC calculation
  CRC check
  Control code
  Destination addressing

Sender addressing

Address recognition*

COUNT calculation

Network receive buffering*

User input buffering*

Control message composition
  Data message*
  Acknowledgement*
  Token
  Initialize**

User interface*
  Format stripping
  Conversion/translation
  Prompting

Power-up procedure

**indicates BIU function only
*indicates CCU function only

in packets as opposed to a continous stream, some method must be used to separate packets. In addition, each packet may require some sort of data preamble to be used to synchronize bus transceivers, since the bus itself does not supply a clock. A commonly used method of packet separation is IBM's Synchronous Data Link Control (SDLC) [11] format, in which packets are delimited by a unique bit pattern called a flag (typically 01111110). If this format is used, however, data must not be allowed to inadvertently reproduce the flag pattern and thus cause a premature termination of a message. This protection, called "bit stuffing," is accomplished by never allowing six consecutive ones in the data portion of a packet. At transmission, whenever five consecutive ones are encountered in the data stream, a zero is automatically inserted. Upon reception, whenever five consecutive ones are encountered, the next bit is automatically ignored. The SDLC format also includes a sixteen-bit Cyclic Redundancy Check (CRC) code for the purpose of error-checking. The CRC is a value calculated from successive shifting and adding of the output bit stream under a prescribed algorithm. The value is calculated as the message is transmitted and appended before the closing flag. At reception, the value is again calculated from the received data. A match of the transmitted value and the value calculated at reception is a very good indication that the received message has no bit errors.

Each station has its own address which it must remember. Because of the bus structure of the network, each station will receive all data transmitted on the network; however, only some of those messages will require action by that station. A station must be able to distinguish between control messages, sent to it and all other stations for the purpose of network maintenance, and

data messages, which may or may not be addressed to it. If a station receives a data message which is addressed to it, it must send an acknowledgement to the sending station and forward the message to the user; otherwise, the message may be discarded or ignored. In addition, each station must keep the current value of COUNT and compare it against its address for transmission clearance. When the station is able to transmit, it must prepare messages in the proper format, including flags, CRC, sender and destination addresses, and designation of the packet as a data message.

Another fundamental requirement is buffering. Because of the transparency requirement, the station never knows when it will receive data from either the bus or the user. When the user transfers a message to the BIU, it must be stored until the station has clearance to transmit on the bus. When the station receives a data message addressed to it from the bus, it must store it until such a time when the user is ready to receive it. Most importantly, the station cannot ignore the bus while receiving or transmitting to the user, or vice-versa, since part of a message might be lost during this dead time. This requirement forces either two separate buffers or capability for very high speed transfers which can be accomplished in less than one bit period of the bus or user's input rate.

To this point, nothing has been said of the form of the exchange between station and user. One of the network's design goals is to be very flexible in this regard, supporting different types of exchange serial, parallel, IEEE-488, etc.), different speeds, and different levels of intelligence in the user interface. A person at a terminal will need to be prompted for information, such as the message and its destination, while a computer or data source is likely to supply that function automatically. One requirement is the same for all, however: before the

message is forwarded to the user, all formatting and any other information used in network management must be stripped off. Beyond that requirement, the station must maintain constant attention to user input by managing and monitoring the appropriate input circuitry.

Finally, each unit must have some standard procedure for initialization after power is turned on. This is necessary to prevent inappropriate operation during power fluctuation or immediately after addition of a station to the network. Some part of the station must have capability to sense power-up and force all other parts of the unit to follow a prescribed procedure until the station has begun regular operation on the network.

B. Matching Functions to Circuitry

The level of complexity implied by the necessary functions indicated the need for a microprocessor or even a multiprocessor environment for the bus units. The appropriate equipment for programming and debugging Intel processors was readily available, making Intel a preferred source. The procedure used for fleshing out the chip usage was to start with the basic microprocessor configuration of processor, Programmable Read Only Memory (PROM) for program storage, Random Access Memory (RAM) for variable storage, and then to add appropriate peripherals.

The choice of a central processor was largely influenced by availability and ease of programming. The choice was between 8085 family processors and 8048 processors, the 8086 being too new and too expensive at the time. While the 8048 family offered single chip ROM (1-2K bytes) and RAM (64 bytes) [12], the 8085 was considered to have a more powerful instruction set and a wider range of peripherals [13]. It seemed clear that more RAM would be needed than 64 bytes for appropriate buffering; thus the

single-chip advantage of the 8048 seemed less attractive. Finally, the 8085 system was seen as the most easily expanded; thus the choice went to the 8085 processor with an Intel 2716 (2K EPROM) for program storage.

The use of variable memory is a more subtle problem. In general, RAM is inexpensive and easily used; for most small applications (< 32K), static RAM is available for less than $10 per kilobyte. The problem is not in expense or ease of use but in efficient access of memory. The most straightforward design of the network would likely dedicate separate banks of memory to user and bus buffers; however, in such a system, an inordinate amount of time would be spent in data flow through the central processor. To minimize the need for central processor intervention (and thus minimize many processing delays), memory for the system was designed as a shared resource for all components, accessible directly by the central processor and available to all peripherals through a Direct Memory Access (DMA) handler, the Intel 8257. This design casts the central processor as a scheduler and manager, and not as a data pipe. Data is stored and retrieved by the peripherals under the direction of the 8085, with only enough information needed to describe the data (length, starting address, type) actually passing through the 8085.

The size of the buffer memory was calculated based on anticipated performance and utilization. Since the network's data rate would always be greater than the transfer rate between the BIU and user, there would always be some possibility for lost data through buffer overflow. In fact, this problem is one of the most serious faults of the network protocol when the user is a slow device (such as a human at a terminal) which may take a long time to complete a message for the system. Since the protocol does not allow the user to be interrupted with output while

composing a message, there is a very real possibility that data will be lost. However, the larger the buffer, the greater the ability to accomodate for speed mismatches. In real use, it was projected that the user would be able to input data at least at half the rate that it was being sent to him; thus the memory was set at 1K bytes (four times the maximum message length allowed, allowing for storage of two user messages and two network messages of maximum length, or any combination of smaller messages). The network buffer was allocated slightly more memory space than the user buffer. The partitioning of memory will be more fully discussed in part C of this section.

The advantages of SDLC formatting suggested use of a dedicated chip to perform the formatting. At the time that the prototype network was being designed, Intel offered only one chip capable of SDLC formatting, the Intel 8273. Other manufacturers had preliminary versions of similar chips, but the combination of guaranteed compatibility and easy availability favored the choice of the 8273. Among the functions offered by the 8273 were automatic flag appending, bit-stuffing, CRC generation, and address recognition. The 8273 could be programmed with two addresses for recognition, in this application, one address was the station's unique address, the other was a broadcast address, used to send all stations control messages. Alternatively, for the CCU, the 8273 could be programmed to receive all messages, regardless of destination address. Two channels of DMA were available, one for transmit data and one for receive data; these were compatible with the 8257 DMA handler. In the DMA mode, the central processor was required only to command the 8273 with receive or transmit commands and message parameters. The data message would be automatically accessed through the DMA handler, and the processor notified at completion. The chip did

impose some restrictions, most notably the limit of 64 kilobits per second as the data rate. In addition, the chip required continual attention from a central processor, being able to free-run only from command to successful reception or transmission. These limitations, however, seemed minor in comparison to the savings afforded by the functions offered.

The last major component of the BIU was the user interface. Based on the protocol and the rest of the design, there were three basic requirements for this component to meet: it must be flexible (e.g. programmable), it must be capable of free-running, at least for periods of time, and it must have capability for short term, small scale storage. The first requirement is one of the original design goals, but the second two are direct implications of the network protocol and design. An explanation of the operation of the DMA handler is needed for background. Since memory is the direct resource of the 8085, the 8085's data and address lines are hardwired to the data and address lines of the memory. The DMA handler also has capability to manipulate the memory address lines, but only during its operation. After the DMA handler has been set by the 8085, it may receive a request from a peripheral for memory access. The DMA handler requests use of the address lines by setting the 8085's HOLD input. When the 8085 has completed its current operation, it acknowledges the request by allowing the address lines to float and ceasing operation until HOLD is released. The DMA handler monitors the transfer between peripheral and memory, and does not release HOLD until the transfer is complete. Thus, during the time that a DMA transfer is being accomplished, the 8085's operation is suspended. In the worst possible case, data might be coming in from the user at the same time the DMA controller was servicing input from the network. As

the data rate of the user approaches the data rate of the network, the potential for lost data increases if the 8085 is required to obtain all data from the user. Prudent practice suggests that the user interface be allowed to run independently of the 8085 and contain enough intelligence and storage capability to allow high-speed transfers to the central memory. For these reasons, it was decided to use a slave microprocessor, the Intel 8741A, for the user interface.

The 8741A is a member of the 8048 family of microprocessors, being a modified version of the 8748. The chief difference between the 8748 and the 8741A is that the external data/address bus on the 8748 is a data bus used only for communication with a master processor on the 8741A. Like the 8748, the 8741A has 1K of program memory and 64 bytes of RAM. Unlike the 8748, the 8741A can be configured to operate with a DMA handler. Combined with the capability of 16 pins of I/O and onboard timing, the 8741A is suited to almost any kind of interface. Perhaps the biggest advantage to the design of the prototype network is that by concentrating the user interface in the 8741A, the user interface may be changed by altering a single chip, without altering the basic operation of the network interface. The user interface, then, is a module consisting of the 8741A and its necessary line-driving or conversion peripherals. The basic hardware layout for the network is shown in Figure IV.1, and the final assignment of functions is shown in Table IV.2.

C. Interconnection of Chips

The interconnection of chips involves not only the specification of wiring diagrams but the protocol for exchange of data between the basic functional blocks of the bus units. This protocol takes the form of establishing

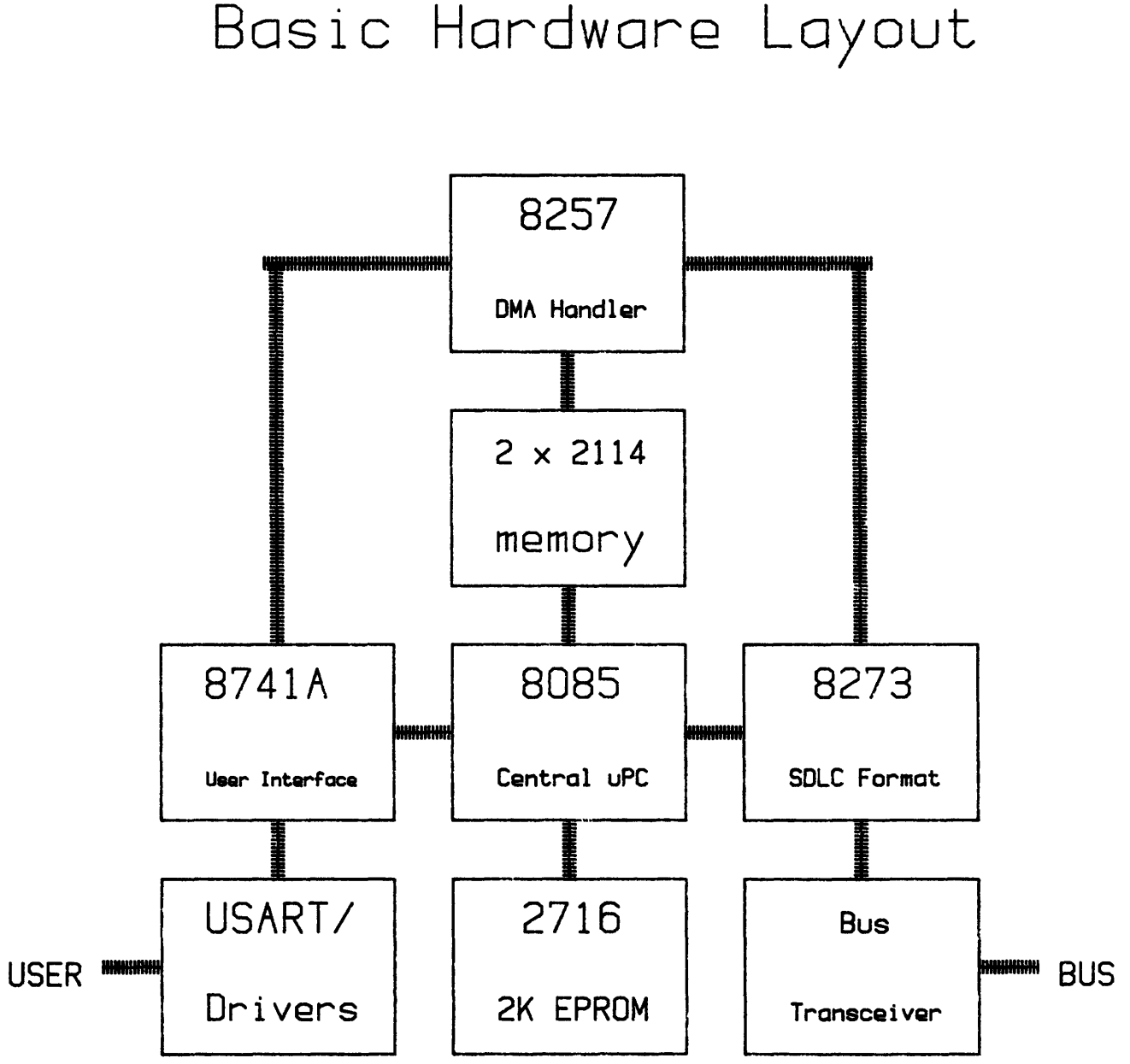


Figure IV.1: Basic Hardware Layout

Table IV.2 Necessary Functions for Network Units

FUNCTION	PERFORMED BY
Bus transceiver management	8273
Data input/output	
Transceiver clock	
Control line management	
SDLC formatting	8273
Flag appending	
Bit stuffing	
CRC calculation	
CRC check	
Control code	
Destination addressing	
Sender addressing	8085
Address recognition*	8273
COUNT calculation	8085
Network receive buffering*	8273 to memory
User input buffering*	8741A and memory
Control message composition	8085
Data message*	
Acknowledgement*	
Token	
Initialize**	
User interface*	
Format stripping	8273
Conversion/translation	8741A
Prompting	8741A
Power-up procedure	8085

*indicates BIU function only
**indicates CCU function only

standard messages and format, and establishing priorities for interrupts and DMA channel usage. The first step in specification of the interconnection is to determine the control and data needs of each station component. Next, addresses must be stipulated for all necessary memory and I/O locations, forming the station's memory map. Finally, a rough flowchart is prepared, from which the actual software can be derived.

In general, each major component in the station will have the following: a chip select input used for enabling operation, address and data lines for transfer of data, a clock input or crystal input for onboard clock generation, registers for indication of status or commands, and a reset input which can be used to force the chip into a predefined state. Because the 8085 is designed always to be the central (or sole) processor, it does not have a chip select input. Operation of the 8085 may only be delayed by the HOLD input, used by the DMA handler in this design. Because of its central role, the 8085 has clock and reset outputs as well as inputs. These outputs can be routed to the various peripheral chips. Finally, its orientation is to send commands and read status, which is just the opposite of the other peripherals which send status and read commands. The clock for the 8085 is derived from a crystal input and the reset input is connected to a switch in parallel with an RC network which will pull reset true shortly after power-up.

Some of the connection for the 8257 DMA handler has already been discussed; the 8257 shares the address and data lines of the 8085 by use of the 8085's HOLD input. In addition, the 8257 requires commands from the 8085 before any action may be taken. The mode register must be set to establish the activity of each of the four available DMA channels and the type of transfers to be made. Each

channel has a set of four registers which must be set, corresponding to two-byte values for starting address of the transfer and the number of bytes to be transferred. The higher order byte of the number to transfer register also contains two bits which indicate whether the transfer is written to or read from memory. Finally, a status register is available for reading by the central processor.

Of four possible channels, three are used in the prototype station design. Each channel has two control lines, DMA request (DRQ) and DMA acknowledgement (DACK), which are used to control transfer on an active channel. The peripheral sets DRQ to initiate transfer. When the DMA handler is ready with valid data (i.e. the handler has control of the data/address lines) it responds by setting DACK. The transfer is completed by normal operation of the read/write lines associated with the data/address lines. Assignment of channel numbers is not a trivial task; in the event of simultaneous request by two or more channels, service is granted by a priority scheme in which the lowest channel number has the highest priority. Thus channel 0 should be assigned to the peripheral least capable of waiting for memory access.

The 8273 SDLC chip is similar in format to the 8257, but requires more commands and provides a wider range of functions. In addition to command and status registers, the 8273 has a parameter register, needed for passing information about commands, and result registers, which provide information about the completion of the last command. The 8273 also provides interrupt outputs which signify the completion of transmission and reception. The commands used in the prototype design are Set Operating Mode, Set Serial I/O Mode, Selective Receive, Transmit Frame, and Abort Transmit. The two mode commands can be set once for all time at power-up. Each requires a byte to

be sent to the command register and a byte to the parameter register. The parameter bytes are used to initiate buffering of address and control parameters, a frame preamble for synchronization, and NRZ data encoding. Selective Receive requires a command and four parameters. The first two parameters are the maximum message length and the second two are the match addresses (one is the station's unique address, the other is the network broadcast address, 0FFH). In Selective Receive, a valid frame will not be saved unless the destination address is one of the match addresses. When a frame with the appropriate address is received, the 8273 saves the address and control bytes for the central processor and stores the message in memory through DMA. An interrupt is generated at completion of the reception, at which time the central processor must read the two saved bytes plus the Receive Result register before any new commands are given. Transmit Frame is similar; it requires a command and four parameters, the first two being the message length and the last two the destination address and control byte. An interrupt is generated at completion of transmission, at which time the Transmit Result register must be read. Abort Transmit is a one-byte command which terminates any transmission in process. Between any two transfers of command and parameter to the 8273, the status register must be checked to insure that the byte has been accepted.

The control byte previously mentioned is not stipulated by the 8273, nor is it used by the chip for any decision-making. It is an eight-bit value which is forwarded to the central processor, making it ideal for the determination of message type. The convention used in the prototype network is that control code 01 represents an initialization message, and that the byte following the control code in the frame is the new value for COUNT.

Control code 02 is the token message; no additional data other than the CRC and flag follows the control code. Control code 03 is the acknowledgement message; the byte following the control code is the address of the station sending the acknowledgment. Control code 04 is for data messages; the first byte following the control is the sender's address, and a message of up to 255 bytes precedes the CRC and closing flag. The format for each of the four different types of messages is shown in Figure IV.2.

The exchange of information between the 8741A and the 8085 is largely left undefined by the chips themselves. The 8741A has a data bus which is dedicated to interaction with a master processor. This bus is buffered and can be loaded at any time. Depending on the sense of an additional control line, the data exchanged is understood to be either data or command. For the purposes of the prototype network, there was a need for four distinct messages from 8085 to 8741A, and another four from 8741A to 8085. The four master-to-slave messages cover the cases when the destination does not respond to a message (code 01), when a message has been received from the network and is ready for forwarding to the user (code 02), when the user interface may proceed to compose a new message into memory (code 03), and when it is necessary for the user interface to wait for another command. In the first two cases, parameters are sent following the command: the no response code is followed by a byte corresponding to the address of the unit which did not respond, and the message received code is followed by a byte which is the length of the message to be forwarded.

The slave-to-master messages are similar, covering the cases when the user interface has completed composing a message to be sent (code 01), when DMA set-up is required for the start of a new message (code 02), when forwarding

Possible SDLC Frames

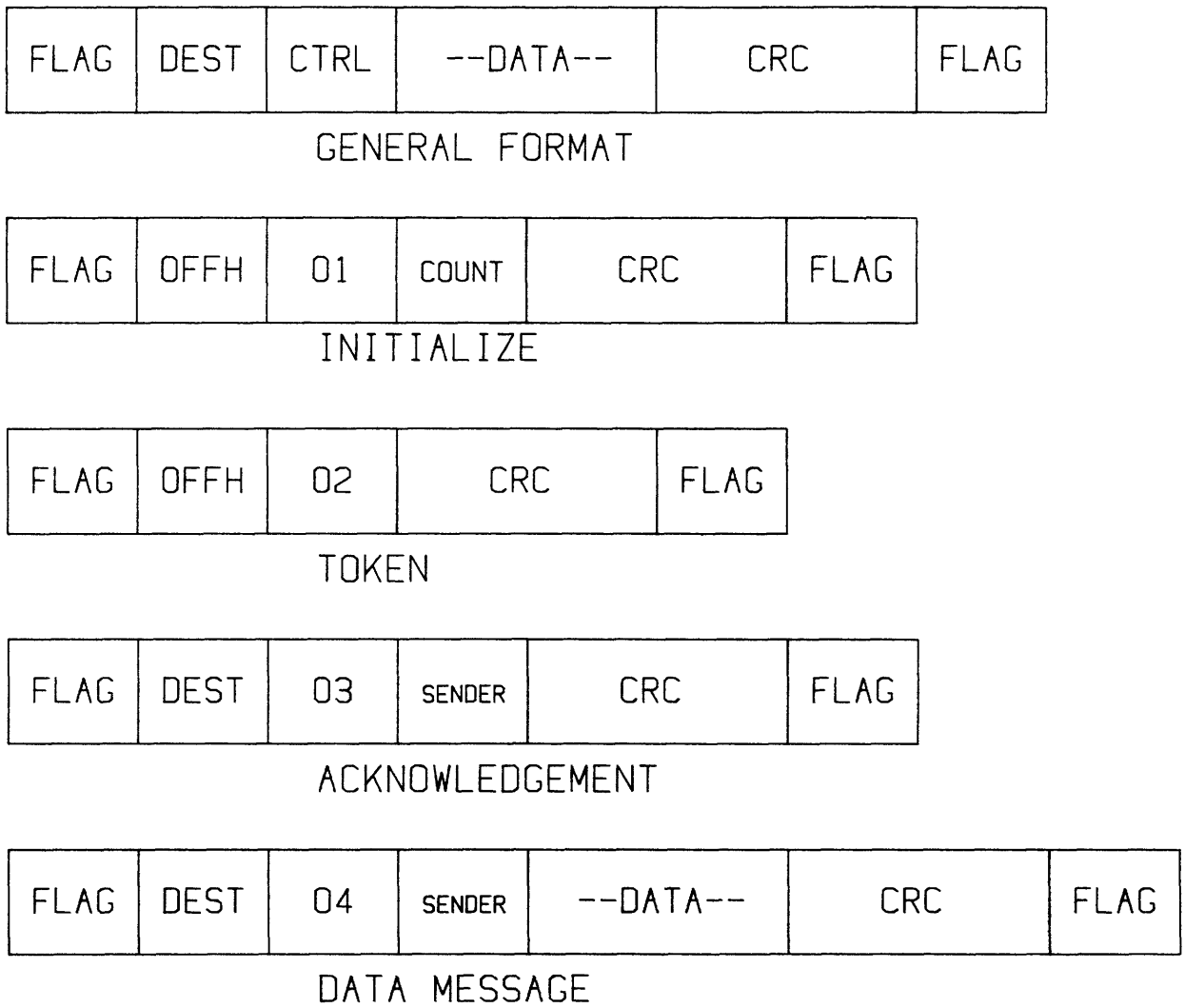


Figure IV.2: Possible SDLC Frames

of a message to the user is completed (code 03), and when a system error requires discontinuation of the current composition process (04). Only the first command supplies any parameters to the 8085; in that case, a byte corresponding to the length of the message is followed by a byte for the destination of the message.

The 8741A is also required to operate some sort of peripheral for communicating with the user. For the prototype network, this peripheral was a Universal Synchronous/Asynchronous Receiver/Transmitter (USART), the Intel 8251. Like the 8273, the 8251 requires mode setting at power-up (7-bit ASCII encoding, asynchronous mode with odd parity, one start bit and one stop bit), and commands for receive and transmit functions. In addition, the user may require prompting during the composition of a message. A stipulation was made that the user would initiate a message by typing a linefeed (ASCII 10) and concluded a message by entering a dollar sign ($). After the dollar sign, the user must enter a two digit value for the destination of the address; the conversion of the address from ASCII to an actual number is handled by the 8741A.

Two additional issues must be discussed before a full description of the hardware design can be given. First, the interrupt structure for the 8085 must be determined. Second, the algorithm for mapping the peripherals must be stipulated. The question of interrupt structure is one of assignment and priority. There is no reason forcing the use of an interrupt structure; all necessary information could be had by continuous monitoring of the status of the individual peripherals. In the interest of speed, however, interrupts are justified. The 8085 allows for three interrupts. Once set by some external source, an interrupt will be serviced following completion of the current instruction. The 8273's receive interrupt was given top

priority, followed by the interrupt output of the 8741A, and the 8273 transmit interrupt. This priority grew out of the faster network data rate and the 8741A's ability to store data on the short term. The transmit interrupt merely indicates availability of a status byte indicating successful transmission, and thus it can be serviced last in the case of conflicting requests.

The choice for address mapping is between so-called "memory-mapped I/O" and "I/O mapped I/O." [14] In memory-mapped I/O systems, there is no distinction made between memory and peripherals as data sources; each has a unique address and can be accessed by any of the memory access instructions. In I/O mapped I/O systems, memory and peripherals are regarded as separate entities; peripherals can only be accessed by use of IN and OUT instructions. The memory-mapped structure has the advantage of allowing more complex arithmetic and logical manipulations of data from peripherals, but requires slightly more time for simple transfers (13 clock cycles as opposed to 10). Because of the need for fast transfers, and the anticipation of little need for arithmetic manipulation of peripheral data, I/O mapped I/O was chosen. In this mapping scheme, each peripheral data source has its own address. An eight bit I/O port is used to manage the peripherals' chip select and control lines. The I/O address is the value the port must output to access a given peripheral. Since some peripherals may have several registers accessed by a set of control lines, it is possible for one peripheral to have several I/O addresses, each corresponding to one particular register. The memory map is shown in Table IV.3 and the I/O map is in Table IV.4.

After an initial wire-wrapped version mysteriously failed to function and proved a near impossible problem in debugging, subsequent versions of the prototype network

were constructed on an SDK-85 microprocessor development board. This allowed easy access to memory and registers for analysis of bugs in development. As a result, the memory map starts at 0800H as opposed to the normal 0, because the development board provides program and memory space that cannot be overwritten. Another revision required was the change of the lowest priority interrupt, required by the development board for a keyboard management function. The 8273 transmit interrupt was disconnected and its function replaced by monitoring of the transmit status register.

The tables and figures that follow contain all necessary information for the actual interconnection and programming of the network stations. The basic circuitry is the same for the CCU and the BIU, with the exception that the CCU contains no user interface, and thus needs no 8741A nor its associated peripherals. The software of the two kinds of stations is similar; however, the CCU uses a general receive command with the 8273, in order that it may monitor all activity on the network. Appendix A details the method used for producing actual assembly language code from the details provided in this section.

Table IV.3 Station Memory Map

ADDRESS	MEMORY CONTENTS
0-07FFH	Monitor programs and memory (PROM)
0800H-0FFFH	Main program memory (PROM)
1000H-10FFH	Buffer for user interface input (RAM, 256 bytes)
1100H-135AH	Buffer for network input (RAM, 603 bytes)
135BH-13BFH	Scratchpad memory (RAM, 101 bytes)
13C0H-13FFH	Program stack (RAM, 64 bytes)

Table IV.4 Station I/O Map

ADDRESS	MNEMONIC	RESIDENT	
10H	IA0DMA	DMA:	Channel 0 address register
11H	IT0DMA		Channel 0 terminal count register
12H	IA1DMA		Channel 1 address register
13H	IT1DMA		Channel 1 terminal count register
14H	IA2DMA		Channel 2 address register
15H	IT2DMA		Channel 2 terminal count register
18H	IMSMDA		Mode setting register
40H	I08273	8273:	Command/Status register
41H	I18273		Parameter/Result register
42H	O08273		Transmit Result register
43H	O18273		Receive Result register
80H	I08741	8741:	Data register
81H	I18741		Command register

Table IV.5 8257 Specifications

Channel 0 (highest priority): 8273 Receive data
Channel 1: 8273 Data for transmission
Channel 2: 8741 receive and transmit data
Channel 3: (lowest priority): not used

Clock input: from 8085 clock output
Reset input: from 8085 reset output

Mode byte:	MSB	Autoload	(off)	1=on
		TC stop	(off)	
		Extended write	(off)	
		Rotate priority	(off)	
		Enable Ch. 3	(off)	
		Enable Ch. 2		
		Enable Ch. 1		
	LSB	Enable Ch. 0		

Commands: Mode set
Set starting address (each channel)
Set terminal count (each channel)

Table IV.6 8273 Specifications

```
Receiver interrupt (RxInt): to 8085 RST 7.5
Transmitter interrupt (TxInt): not used
Receiver DMA (RxDRQ and RxDACK): to 8257 Channel 0
Transmitter DMA (TxDRQ and TxDACK): to 8257 Channel 1

Clock input: from 8085 clock output
Reset input: from 8085 reset output

Modem control lines (RTS, CTS, CD): to bus transceiver
Transmit Clock: from clock generator

Modes:
  One bit delay          (off)   Command 64H
  Data transfer mode     (off)   Command 57H
  Operating mode byte
    MSB HDLC mode        (off)  1=on
        EOP interrupt    (off)
        Early Tx int     (off)
        Buffered mode    (on)
        Preframe sync    (on)
        Flag stream      (off)   Command 91H
                                 Parameter 06H

Commands:
  Selective Receive (for BIU only)
   Command C1H
   Parameter <Buffer size (L)>
   Parameter <Buffer size (H)>
   Parameter <Match address#1>
   Parameter <Match address#2>

  General Receive (for CCU only)
   Command C0H
   Parameter <Buffer size (L)>
   Parameter <Buffer size (H)>

  Transmit Frame
   Command C8H
   Parameter <number of bytes (L)>
   Parameter <number of bytes (H)>
   Parameter <destination address>
   Parameter <control code>

  Abort transmit
   Command CCH

  Abort receive
   Command C5H
```

Table IV.7 8741 Specifications

Data Bus (D0-D7): to 8085 Address/data bus
Chip select: tied low
Clock input: 6.144 MHz crystal
Reset input: 8085 reset output

I/O Port 1: 8251 output lines
I/O Port 2: P10-P12 to 8251 control lines
P24 (OBF) to 8085 RST 6.5
P26, P27 to DMA Channel 2
Tests (T0 and T1): to 8251 RxRDY and TxRDY lines

Modes: none

Commands from 8085:

No response	Command 01H Parameter <station address>
Message for user	Command 02H Parameter <message length>
Begin DMA from user	Command 03H
Hold: no DMA	Command 04H

Messages to 8085:

User message complete	Status=01H Parameter <message length> Parameter <destination address>
Request DMA	Status=02H
Transfer to user done	Status=03H
Abort message buffer	Status=04H

Table IV.8 8251 Specifications

I/O lines (D0-D7): to 8741A Port 1
Control lines: from 8741A Port 2
Data received interrupt (RxRDY): to 8741A Test 0
Transmitter ready interrupt: to 8741A Test 1

Chip select: tied low
Clock input: from 8085 clock output
Reset output: from 8085 reset output

Transmit data (TxD): to user
Receive data (RxD): from user
Transmit clock (TxC): from clock generator
Receive clock (RxC): none (asynchronous operation)

Mode Byte:

Stop bits: 1	D7=0	
	D6=1	
Parity: odd	D5=0	
Enable parity	D4=1	
6-bit characters	D3=1	
	D2=0	
Baud rate factor: 16	D1=1	
	D0=1	Mode byte=5BH

Command byte:

Hunt mode	(off)	0	MSB
Internal reset	(off)	0	
Request to send	(off)	0	
Error reset	(off)	0	
Send break	(off)	0	
Receive enable			
Data Terminal Ready	(off)	0	
Transmit enable			

Table IV.9 8085 Specifications

Interrupts:
RST 7.5 8273 Receive interrupt
RST 6.5 8741A Output Buffer Full flag
RST 5.5 SDK-85 kit board use

Reset: Debounced switch and RC network (for power-up)

Clock: Derived from 6.144 MHz crystal

I/O Mapping: I/O mapped (see memory and I/O maps)

Program memory: 2K bytes (Intel 2716)

Working memory: 1K bytes (two Intel 2114)

DETAILED FLOWCHARTS

The next 21 pages contain the detailed flowcharts for the BIU, CCU, and 8741. They are the basis for the assembly language programming of the network units. In essence, they are an amplification of the flowcharts which appear in Section III. The flowcharts contained here should be consulted only for information regarding implementation of function, while those of Section III provide the clearest picture of the network operation.

Figure IV.3.0 Detailed Flowcharts

BIU Flowchart: Power-up/Reset

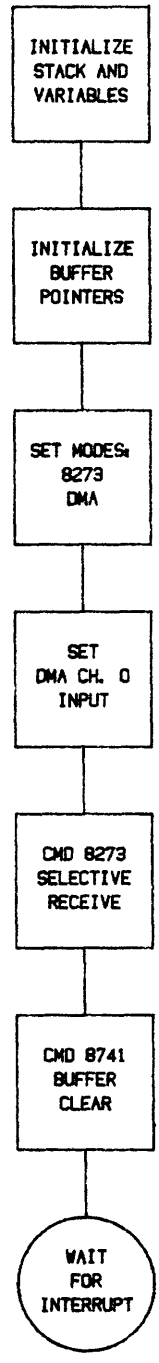


Figure IV.3.1 Detailed Flowchart: BIU

BIU: On RST 6.5 (8741 Interrupt)

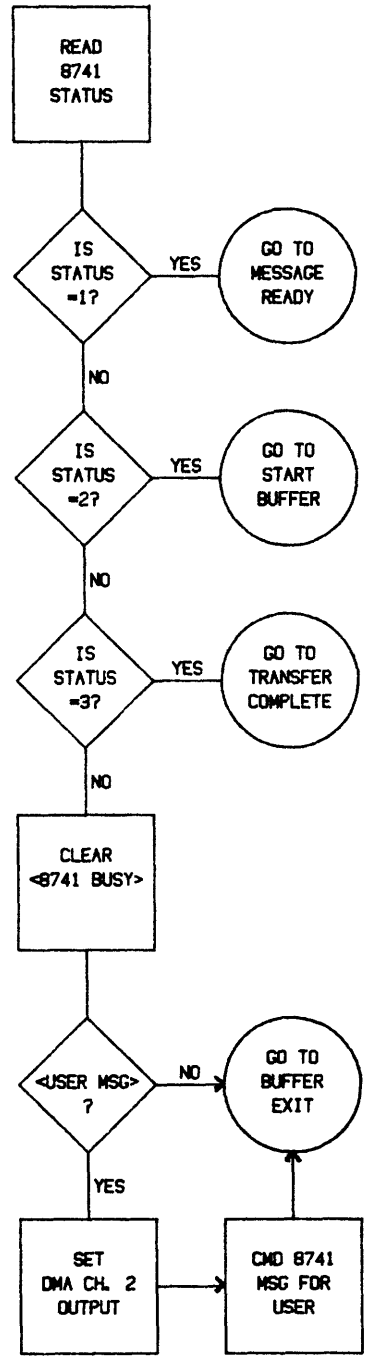


Figure IV.3.2 Detailed Flowchart: BIU

BIU: Message Ready

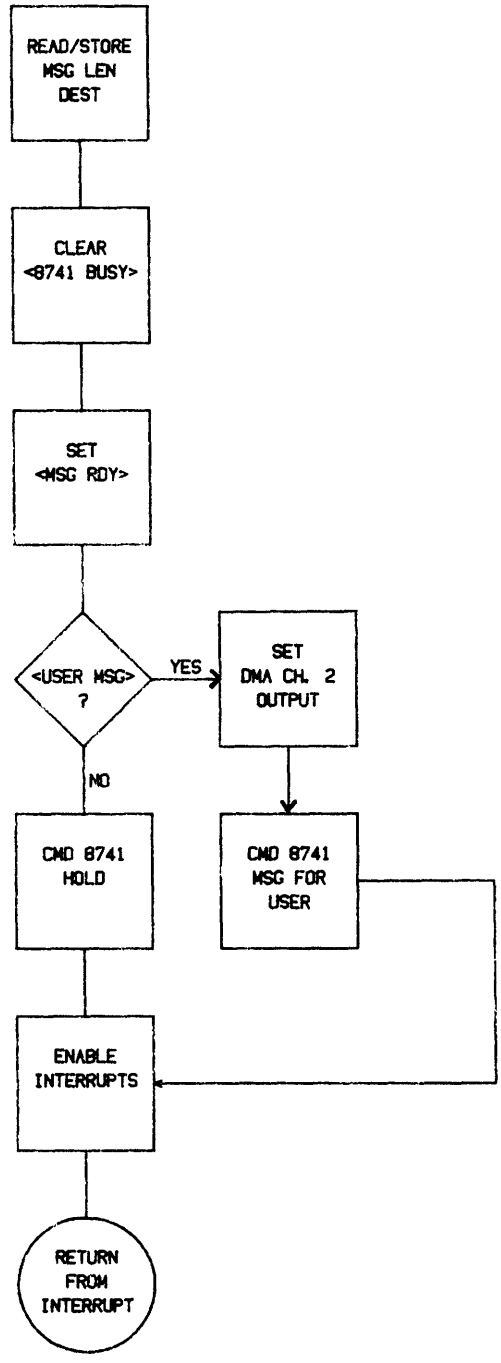


Figure IV.3.3 Detailed Flowchart: BIU

BIU: Start Buffer

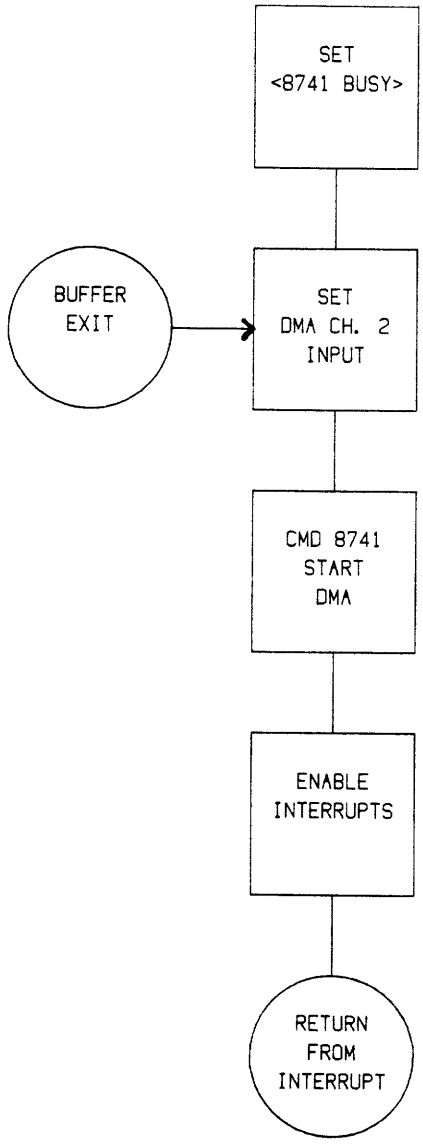


Figure IV.3.4 Detailed Flowchart: BIU

BIU: Transfer Complete

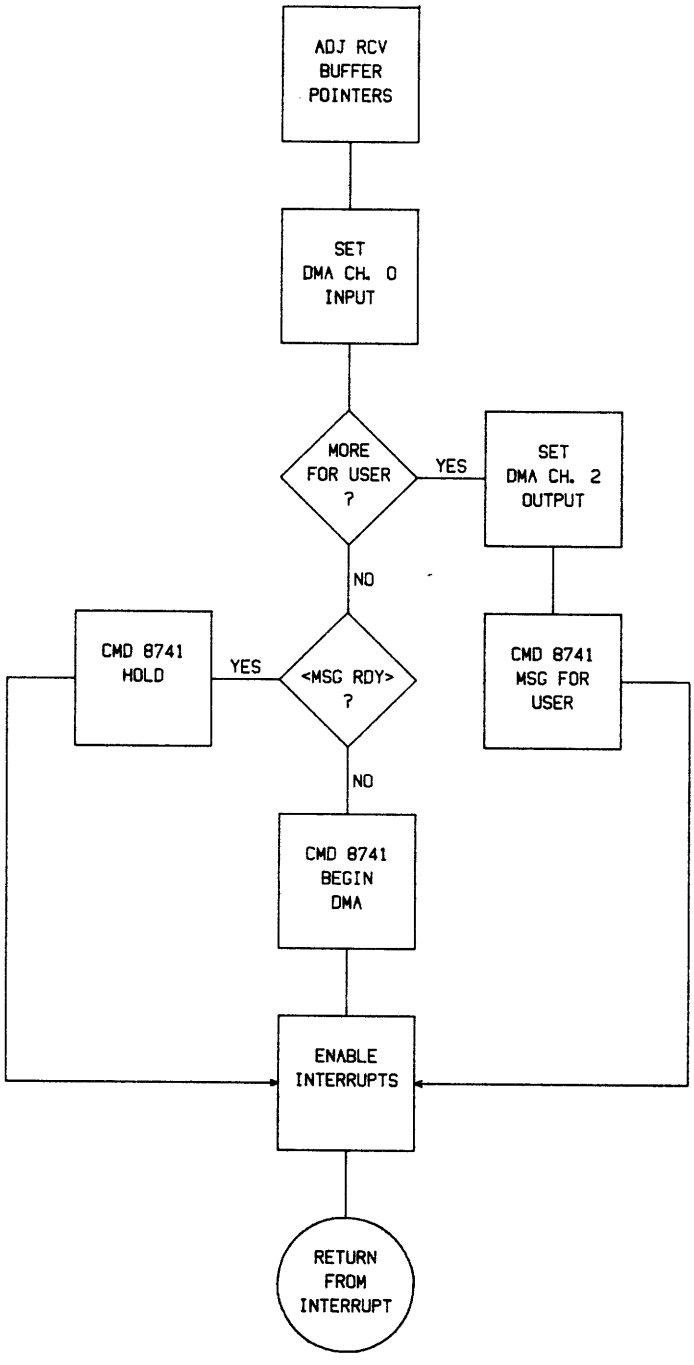


Figure IV.3.4 Detailed Flowchart: BIU

BIU: On RST 7.5 (8273 RxInt)

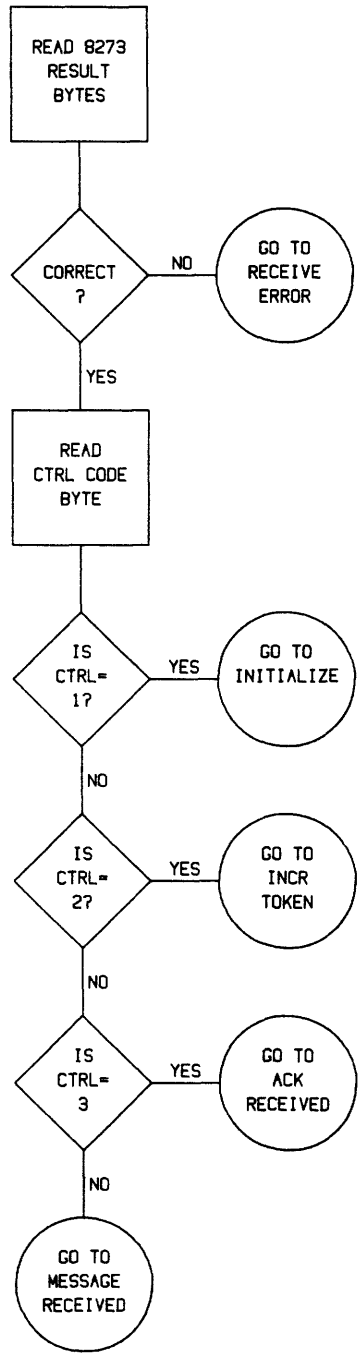


Figure IV.3.5 Detailed Flowchart: BIU

BIU: Receive Error

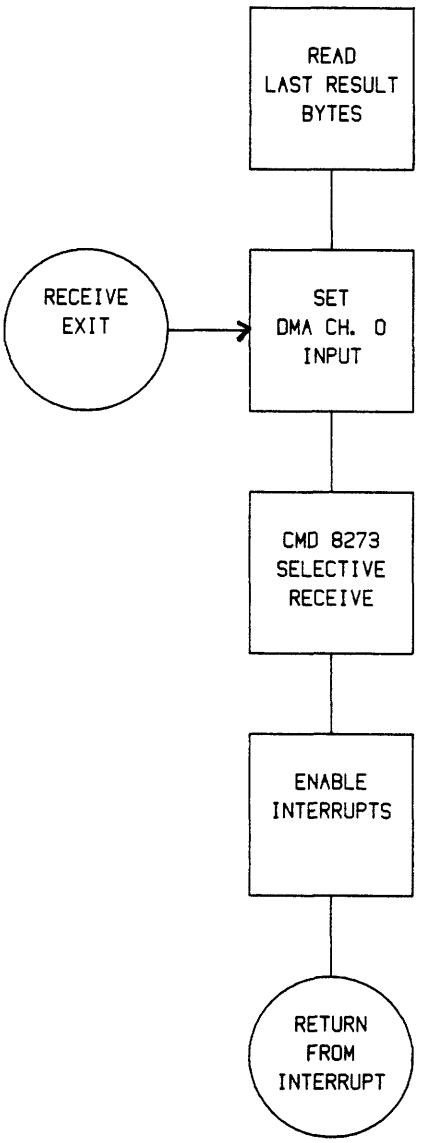


Figure IV.3.6 Detailed Flowchart: BIU

BIU: Initialize

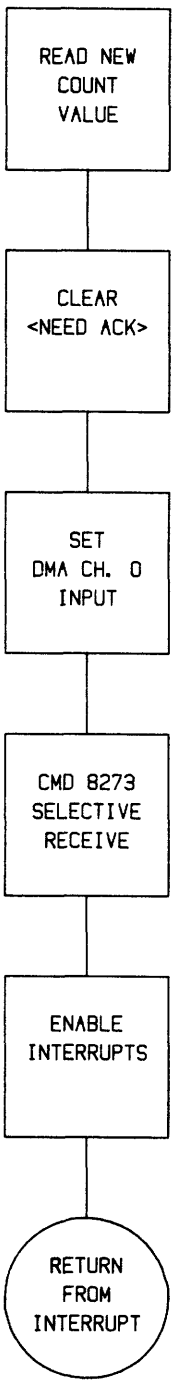


Figure IV.3.7 Detailed Flowchart: BIU

BIU: Incr Token

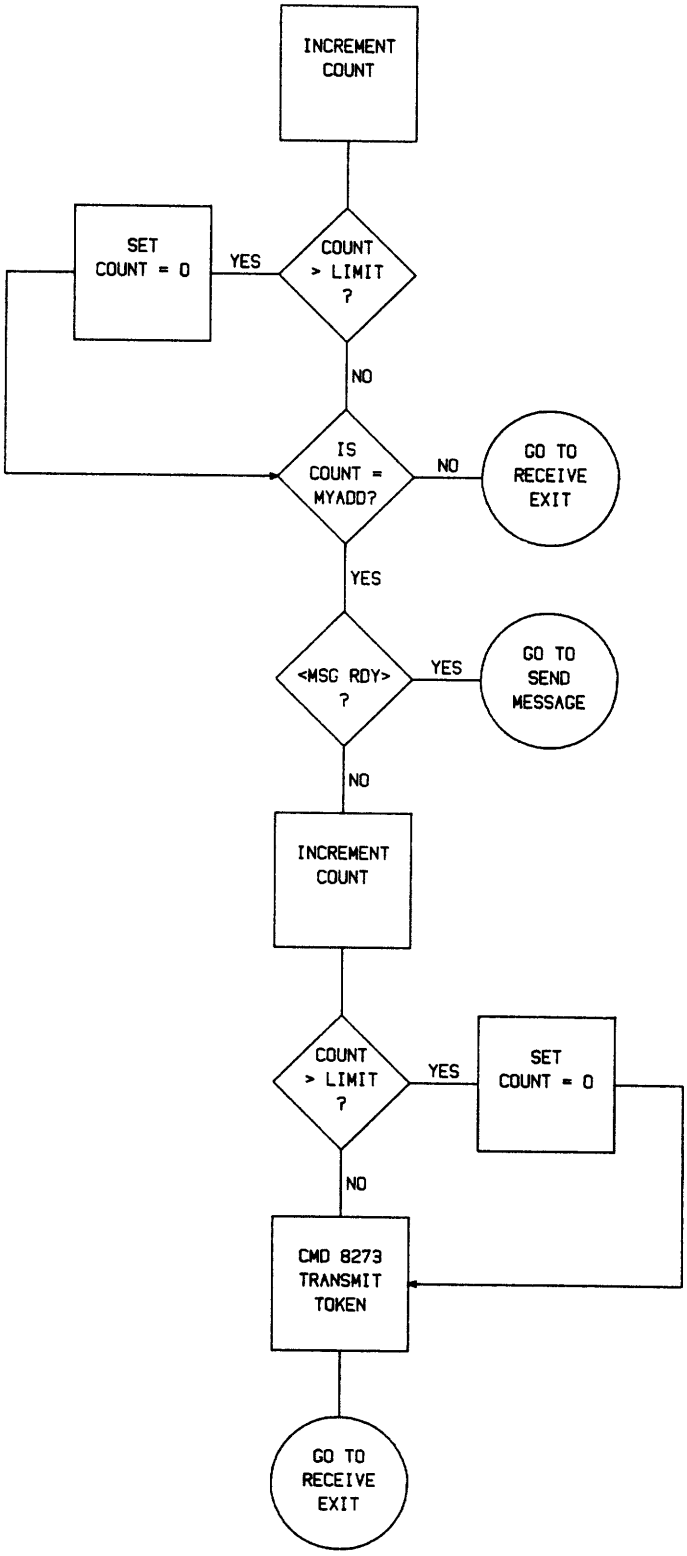


Figure IV.3.8 Detailed Flowchart: BIU

BIU: Send Message

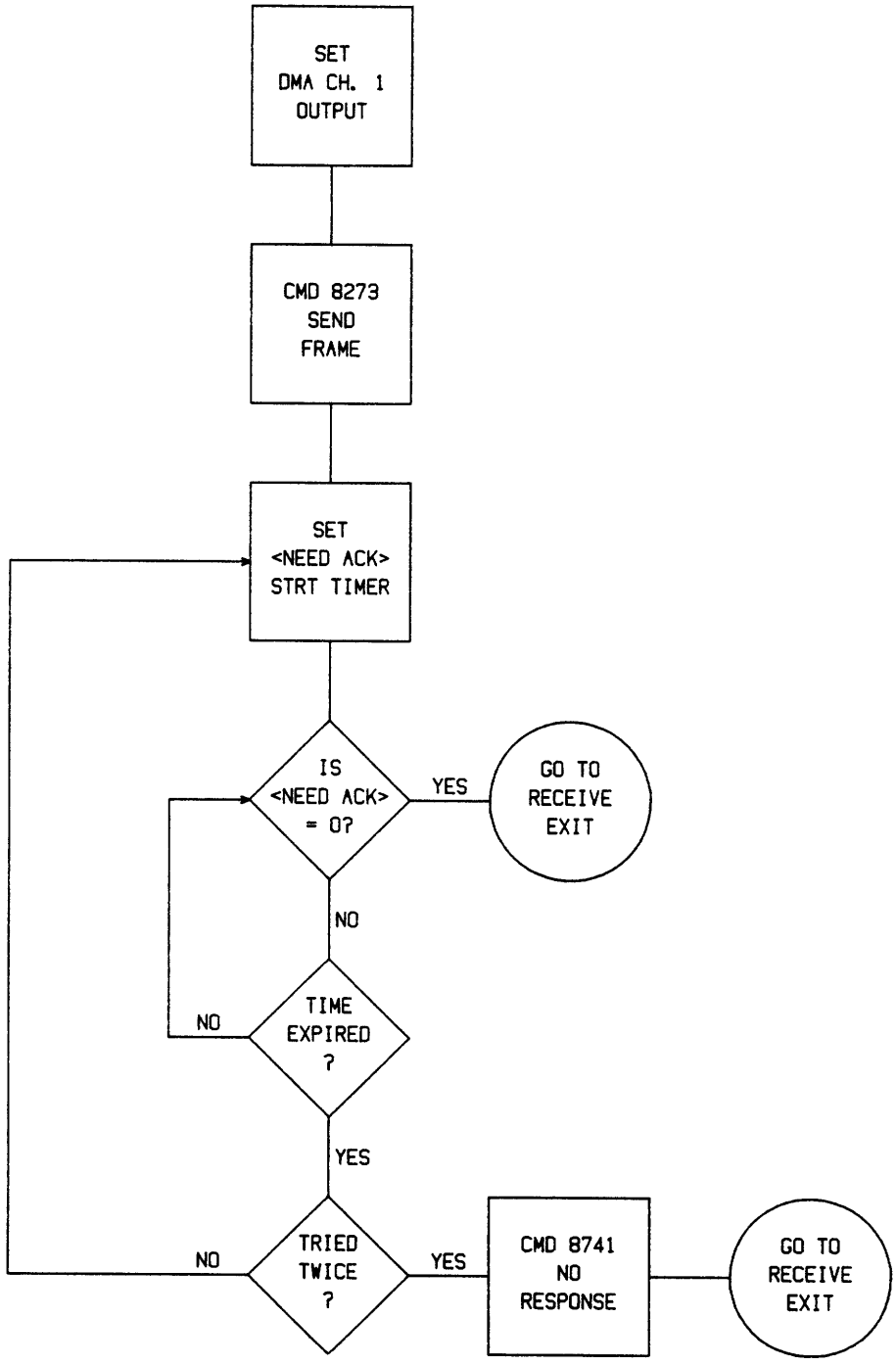


Figure IV.3.9 Detailed Flowchart: BIU

BIU: Ack Received

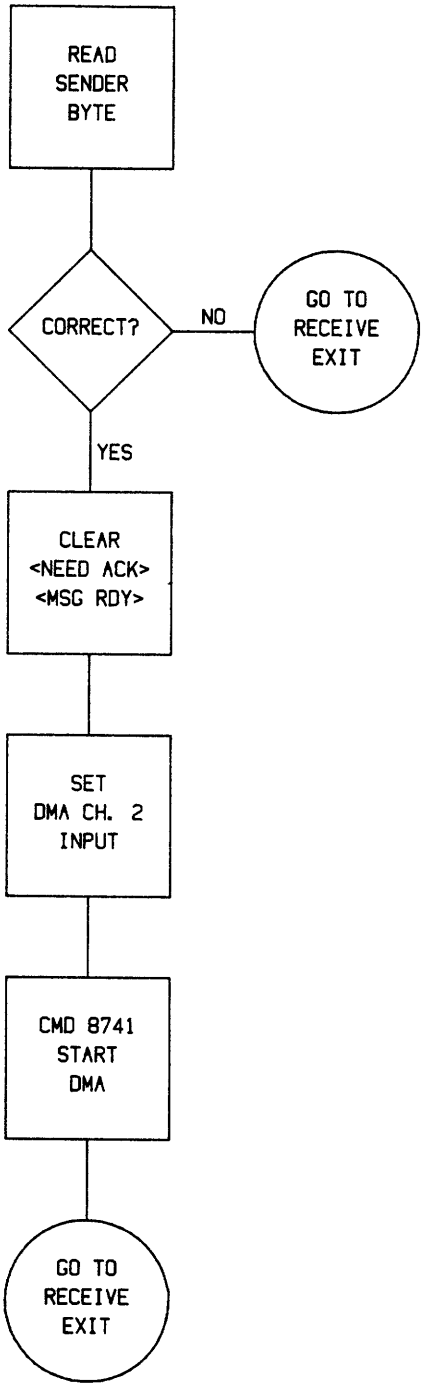


Figure IV.3.10 Detailed Flowchart: BIU

BIU: Message Received

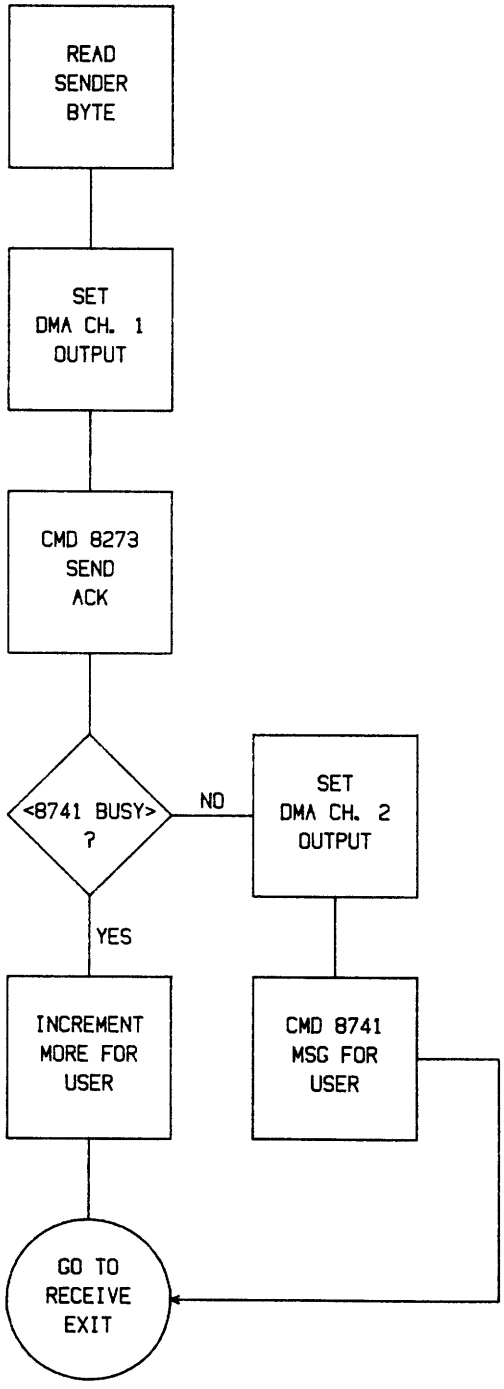


Figure IV.3.11 Detailed Flowchart: BIU

CCU Flowchart: Power-up/Reset

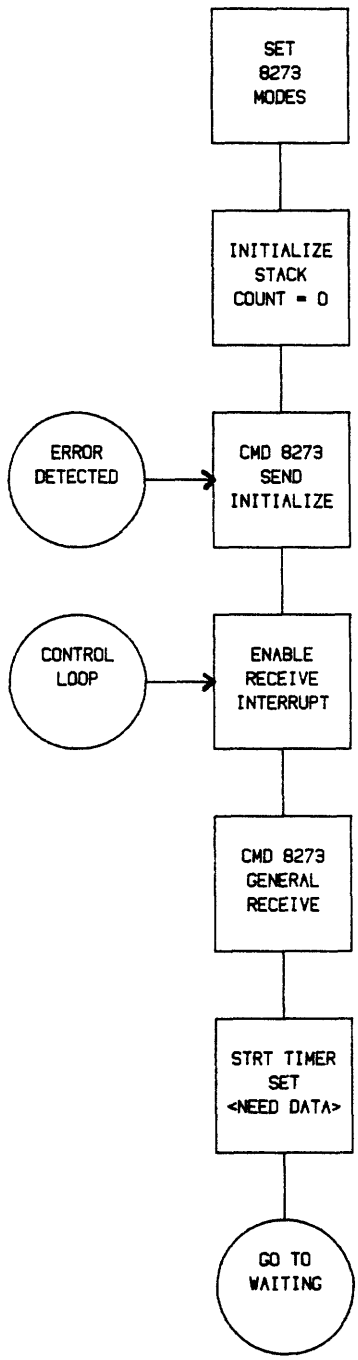


Figure IV.4.1 Detailed Flowchart: CCU

CCU: Waiting

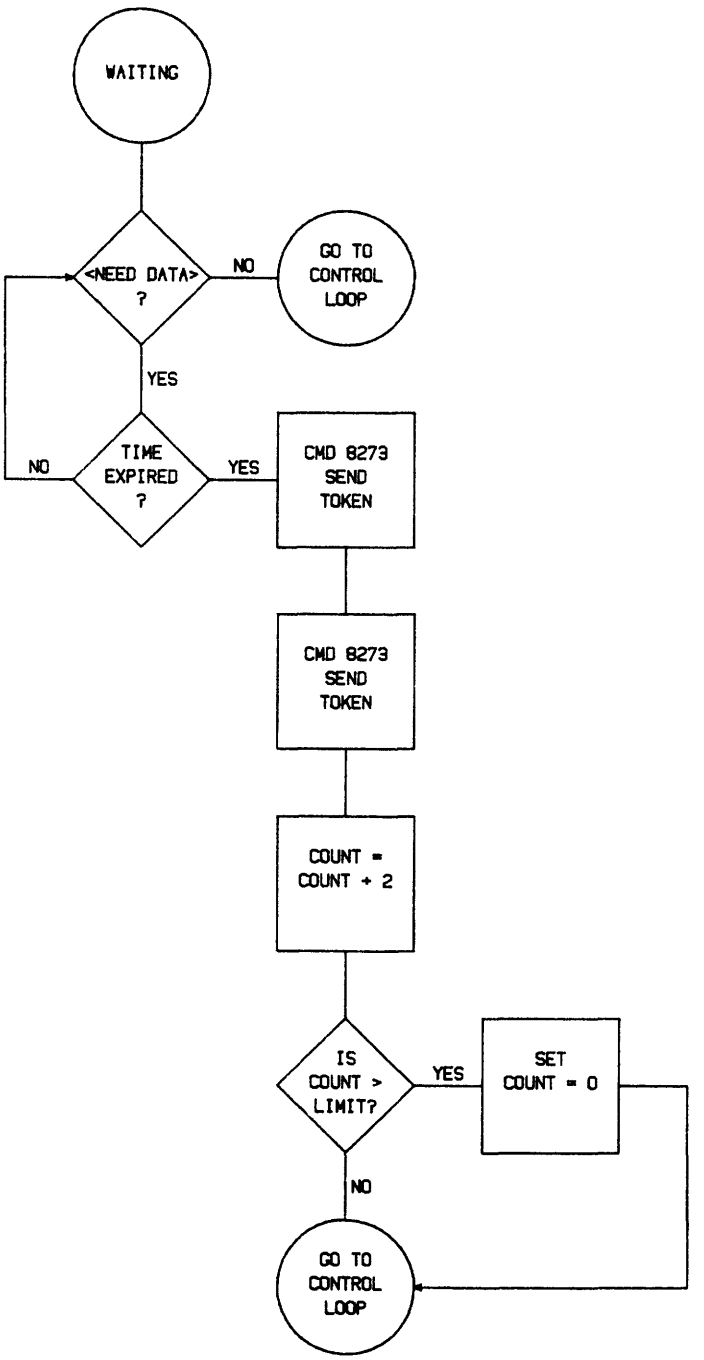


Figure IV.4.2 Detailed Flowchart: CCU

CCU: On RST 7.5 (8273 RxInt)

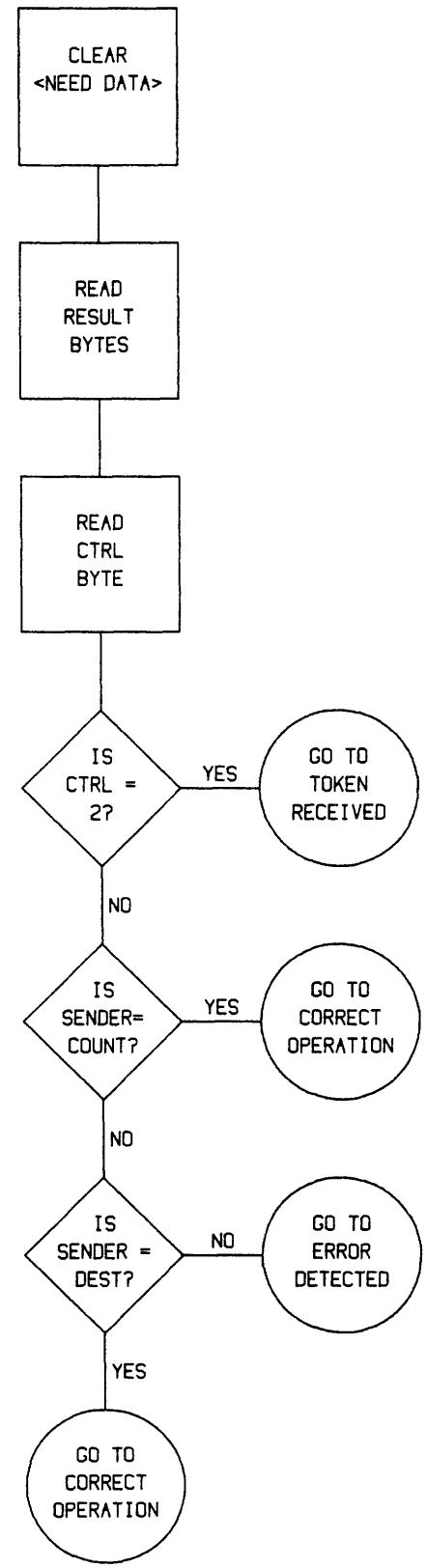


Figure IV.4.3 Detailed Flowchart: CCU

CCU: Token Received

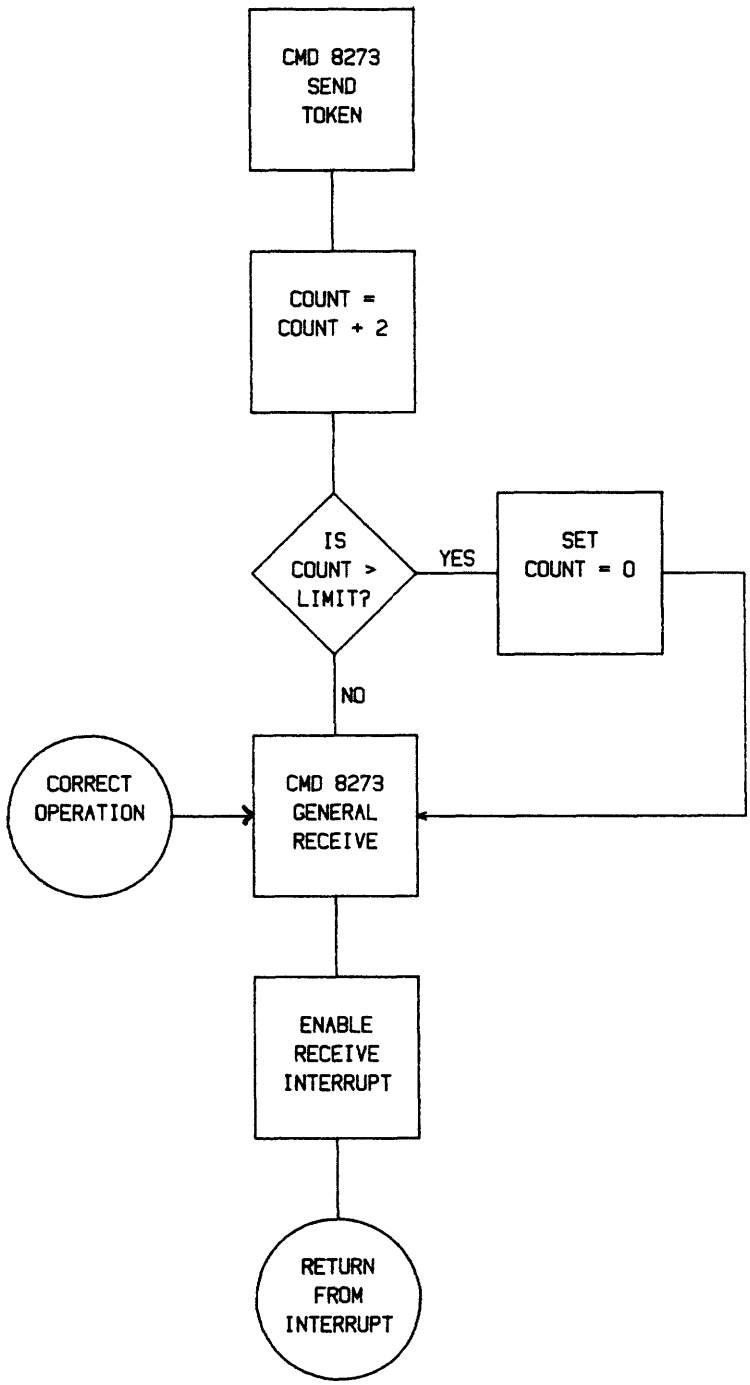


Figure IV.4.4 Detailed Flowchart: CCU

8741 Flowchart: Power-up/Reset

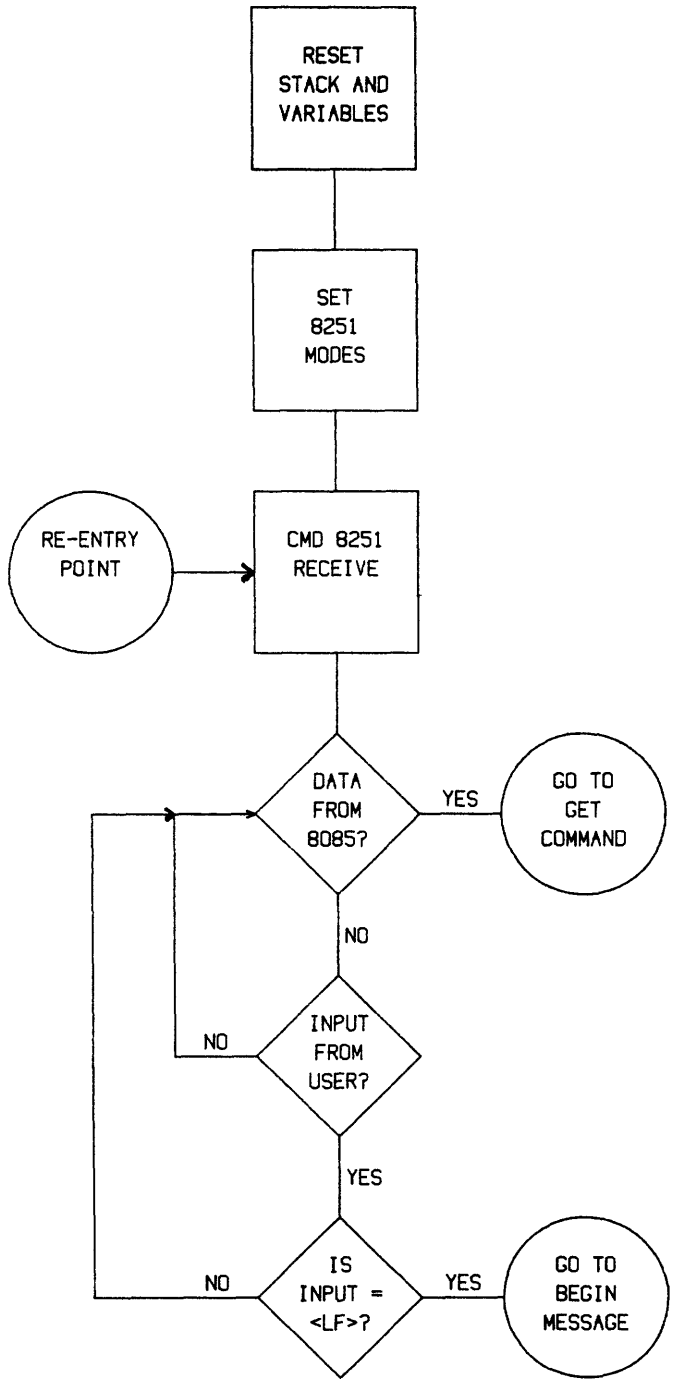


Figure IV.5.1 Detailed Flowchart: 8741

8741: Begin Message

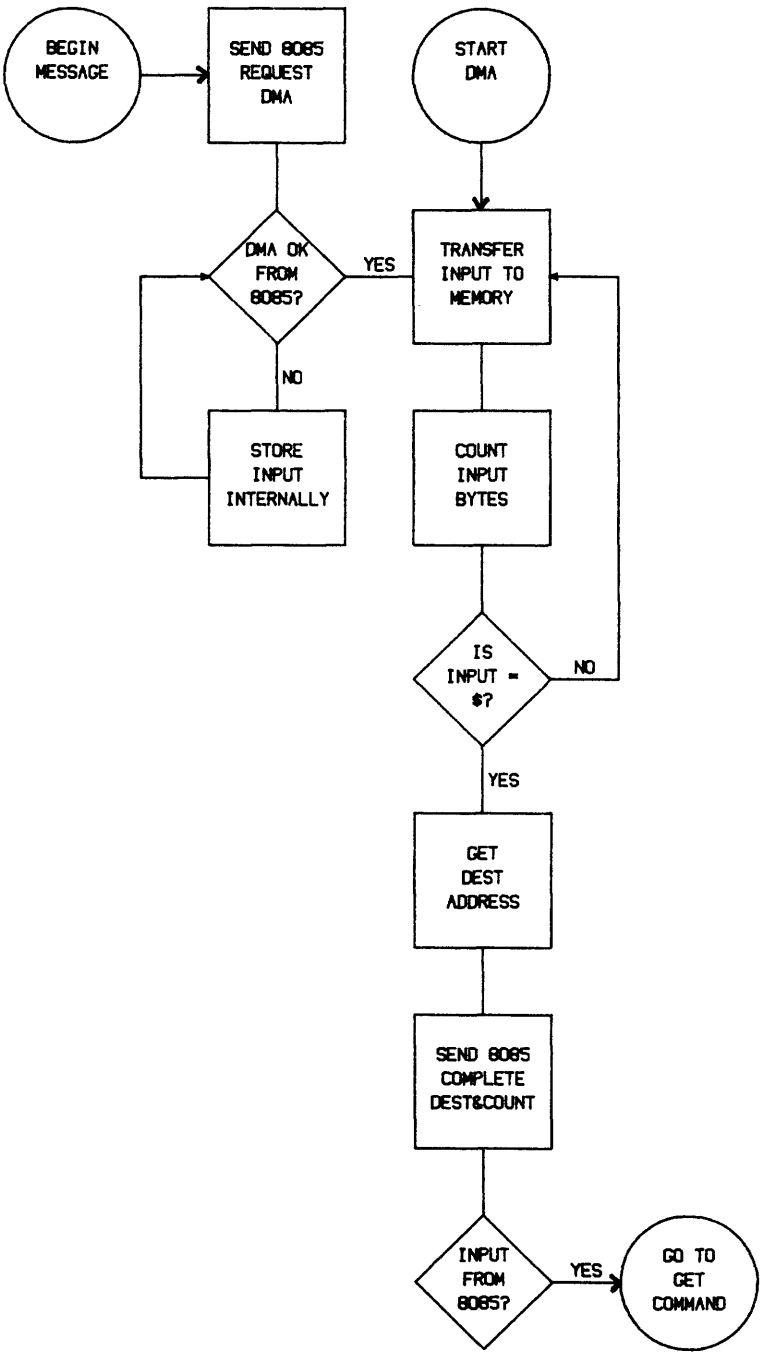


Figure IV.5.2 Detailed Flowchart: 8741

8741: Get Command

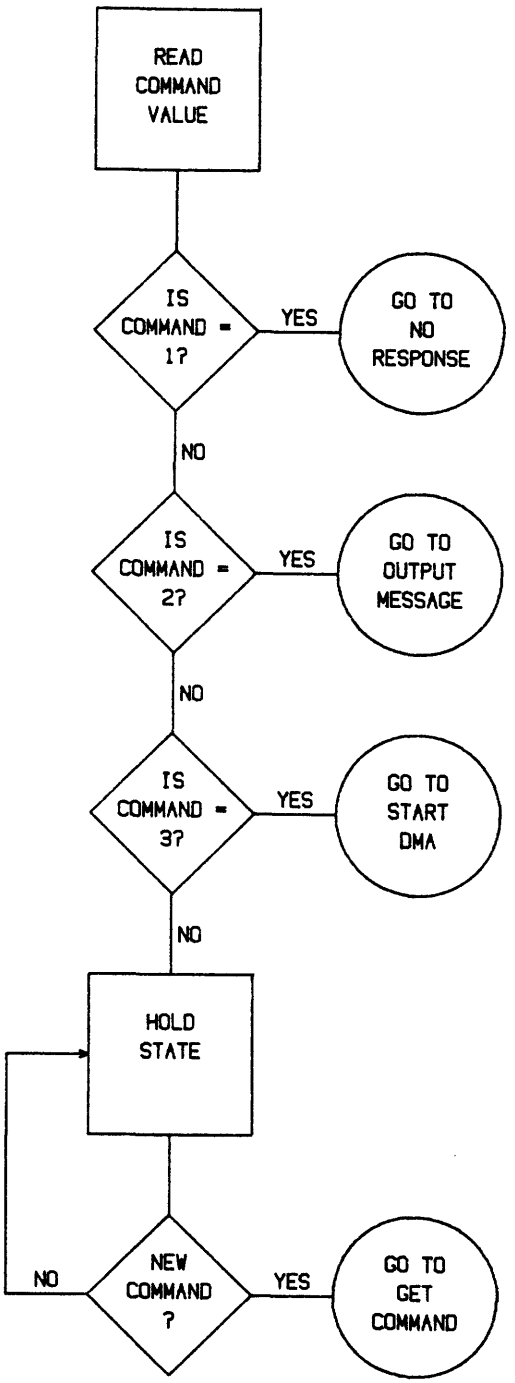


Figure IV.5.3 Detailed Flowchart: 8741

8741: No Response

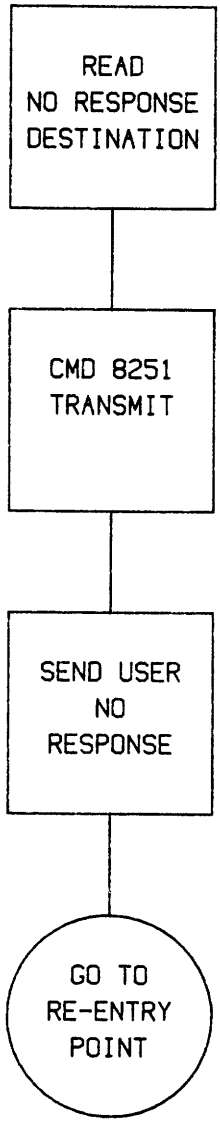


Figure IV.5.4 Detailed Flowchart: 8741

8741: Output Message

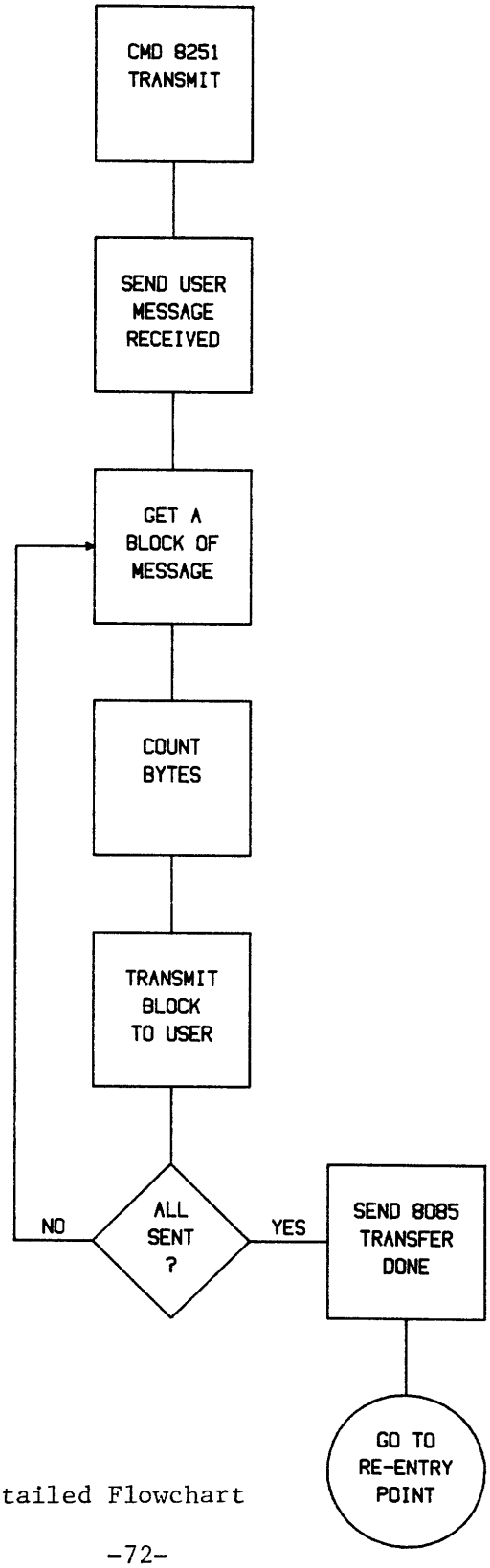


Figure IV.5.5 Detailed Flowchart

V. Analysis of the Prototype Network

A. Definition of Performance Parameters

After a prototype network of four stations had been successfully brought to operation, a quantitative measure of the performance of the system was desired. Primarily, this measure concerned the data-handling capability of the network (a discussion of reliability of the network is contained in the first part of Section VII), which may be characterized by three parameters: access delay, efficiency, and throughput. Part A of this section concludes with a definition of these parameters, part B is devoted to the development of a mathematical model of the network operation, and part C uses this model to project values for the three parameters.

In order to accurately describe the network, the general definitions of the performance parameters must be tailored to the network's unique features. These parameters may not be directly measurable, but once determined, they should give an accurate description of regular operation. For instance, because of the network transparency requirement, the user is not aware of any access delay time. However, if a steady stream of data is fed to a network interface, the delay will have a noticeable effect. The access delay is defined as the time from user completion of a message (including destination address) to the time at which that message is actually transmitted on the network medium. Because of the independent operation of the user and bus interfaces, the access delay is not a constant quantity (consider the difference in delay for the cases where the user completes input of a message just before the station's token COUNT is reached, as opposed to just after). For this reason, coupled with the negative nature of delay, the access delay is considered for the

worst case (i.e. message completed just after COUNT reached). In the case of the prototype network, the access delay corresponds to the amount of time needed to transfer a completed message from the user interface to memory plus the amount of type taken by the network to cycle through all possible values of COUNT. The high-speed nature of the user interface-to-memory transfer means that the COUNT cycle time is the principal component of access delay.

Efficiency is generally defined as the percentage of information transmitted that is "useful." [15] "Useful" is defined in relation to the user, not the network or its internal function. Useful information excludes any data appended to a message for network management purposes, that is, only the actual data message is useful in this sense; whatever other information used for formatting, addressing, error-checking, or transceiver synchronization is considered to be non-useful "overhead." This definition of efficiency will require a subtle change to be fully indicative of the prototype network's efficiency. In a random-access network, there always exists some level of utilization at which all time is used in transmission of data, i.e. there is no "dead time" when there is no data on the network medium. In any deterministic-type network, such as the prototype network, just the opposite is true: there will always be some dead time while a consensus is being reached on which unit has the right to transmit. If efficiency is to reflect the dynamic operation of the network, it must take any dead time into account. Thus the definition of efficiency is altered to be the percentage of time spent in transmission of useful information. The result is that the longer the dead time, the lower efficiency, so that the efficiency may be viewed as a measure of the utilization of the network data rate.

Throughput is the rate of data transfer through the

network. From the argument of the previous paragraph, if that definition is amended to be the rate of useful data transfer, then the throughput is merely the product of the efficiency and the network data rate. This definition is realistic since the useful data is sole concern of of the user; the actualities of data flow inside the network are not important from a performance standpoint. With these definitions, an accurate performance model of the network needs only to describe the timing of network activity.

B. Mathematical Performance Model

Although the user's input data rate to a given BIU is generally much slower than the bus transmission rate, the combination of many users can achieve almost any collective input data rate. Since the network medium is the part of the network through which all messages must pass, it is the basis of the performance model for the network. As was argued previously, the time spent in transfer of a message from user interface to memory is insignificant in comparison to the time spent waiting for authorization to transmit. Thus for low enough network data rates (less than 10M bits per second), the significant performance parameters can be determined from a consideration of message flow on the network medium, ignoring the smaller delays associated with message transfer to and from memory.

The following argument is based on a hypothetical observation of the network medium over a period of time. This observation is made at one end of the bus, called node zero, where the location of each station corresponds to a node. The CCU is assumed to be located at node zero. Besides allowing easier conceptualization, this assumption corresponds to the worst location for the CCU (i.e. the location which maximizes propagation delays). The magnitude of propagation delays is minimized by central

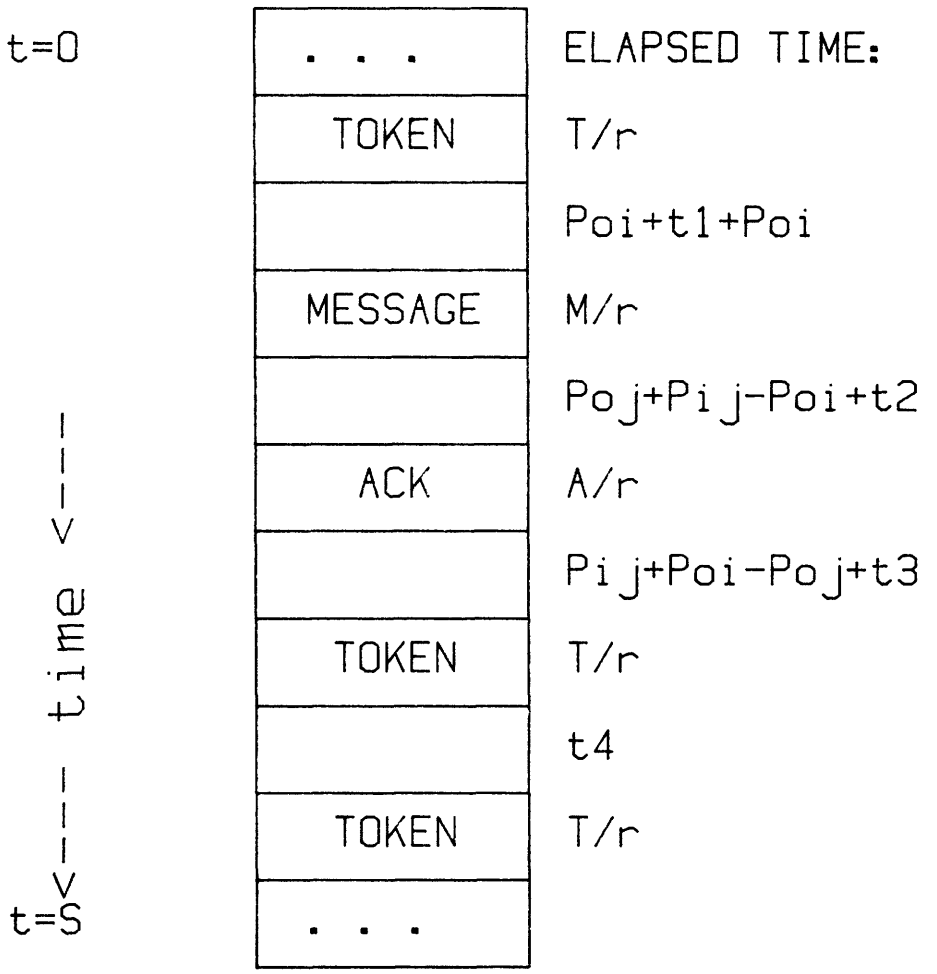
location of the CCU. Finally, the observation is based on the additional assumptions that all BIU's are identical in terms of processing times, that all BIU's are functioning, and that communication on the network is evenly distributed, i.e. over a long enough period of time, any given BIU will send messages to all other BIU's. Other simplifying assumptions will be stated as they are made. Table V.1 shows the meaning of the variables used in the equations and figures.

The model follows from the sequence of events under two different conditions, first when each BIU has a message of length M to send, and second when none of the BIU's has a message to send. Under the first condition, the COUNT cycle time will be maximumized (corresponding to maximum access delay) and under the second, COUNT cycle time will be minimized (minimum access delay).

Figure V.1 shows the sequence of events under the first condition, starting just after the CCU has issued a token message. COUNT is assumed to have the value which will enable station i to transmit. For T/r seconds, the token message is passing node 0. Afterwards there is dead time while the message propagates to node i (Poi), station i receives the token and realizes that it may transmit (t1), and finally the message which station i sends propagates back to node 0 (Poi). For the next M/r seconds, the message is passing node zero, and afterwards another dead time begins. Assuming that the message was sent to station j, this dead time corresponds to the time taken for the message to travel from node i to node j less the time the message took to get to node zero (Pij - Poi), plus the processing time before station j to sends the acknowledgement message (t2) and the time taken for the acknowledgement message to reach node 0 (Poj). For the next A/r seconds, the acknowledgement message passes node

BUS ACTIVITY VIEWED AT CCU NODE (node 0)

BIU i sends message to BIU j



$$S = 2Poi + 2Pij + (2T+A+M)/r + t1 + t2 + t3 + t4$$

Figure V.1: Bus Activity (with messages)

Table V.1 Definition of Variables

variable	meaning (units)
T	token frame length (bits)
A	acknowledge frame length (bits)
M	message frame length (bits)
L	minimum loop time per BIU (seconds)
S	maximum loop time per BIU (seconds)
r	bus data rate (bits per second)
Pij	propagation time from node i to node j (seconds)
tk	kth processing time (seconds)

Where each tk corresponds to the time from a given input received until a given output is issued by a particular kind of unit:

tk	input received	to	output sent	unit
t1	token (COUNT=MYADDR-1)		message	BIU
t2	message		acknowledge	BIU
t3	acknowledge		token	BIU
t4	token		token	CCU
t5	token (COUNT=MYADDR-1)		token	BIU
t6	token (COUNT=LIMIT)		token	CCU

0, followed by a dead time corresponding to the propagation time of the acknowledgement to node i less the propagation time of the acknowledgement to node 0 (Pij-Poj), plus the time taken after reception of the acknowledgement before station i sends the token message (t3), and the time for the token message to reach node 0 (Poi). For T/r seconds the token message passes the node, followed by a processing delay (t4) before the CCU issues the token message which will allow station i+1 to transmit. The total time taken for the single station S, is given by:

$$S = 2Poi + 2Pij + (2T + A + M)/r + t1 + t2 + t3 + t4.$$

The process is repeated for N BIU's in completing one cycle of COUNT, which makes the total maximum loop time, ST:

$$ST = 2\sum_{i=1;i\#j}^{i=N}(Poi + Pij) + N(2T + A + M)/r + N(t1+t2+t3+t4) + t6-t4.$$

The t6-t4 term is added to account for the fact that after COUNT has been cycled, COUNT=LIMIT and must be set to zero before the next cycle.

Figure V.2 shows the second case, a simplified version of the first where there are no messages to be sent. The processing time is t5 for all stations, the time between reception of the token and transmission of the token when no messages are ready. All messages propagate between CCU (node 0) and the ith BIU and thus the loop time per BIU for this case (minimum access delay), L, is given by:

$$L = 2Poi + 2T/r + t4 + t5.$$

Similarly, for the total group of N BIU's, the total loop

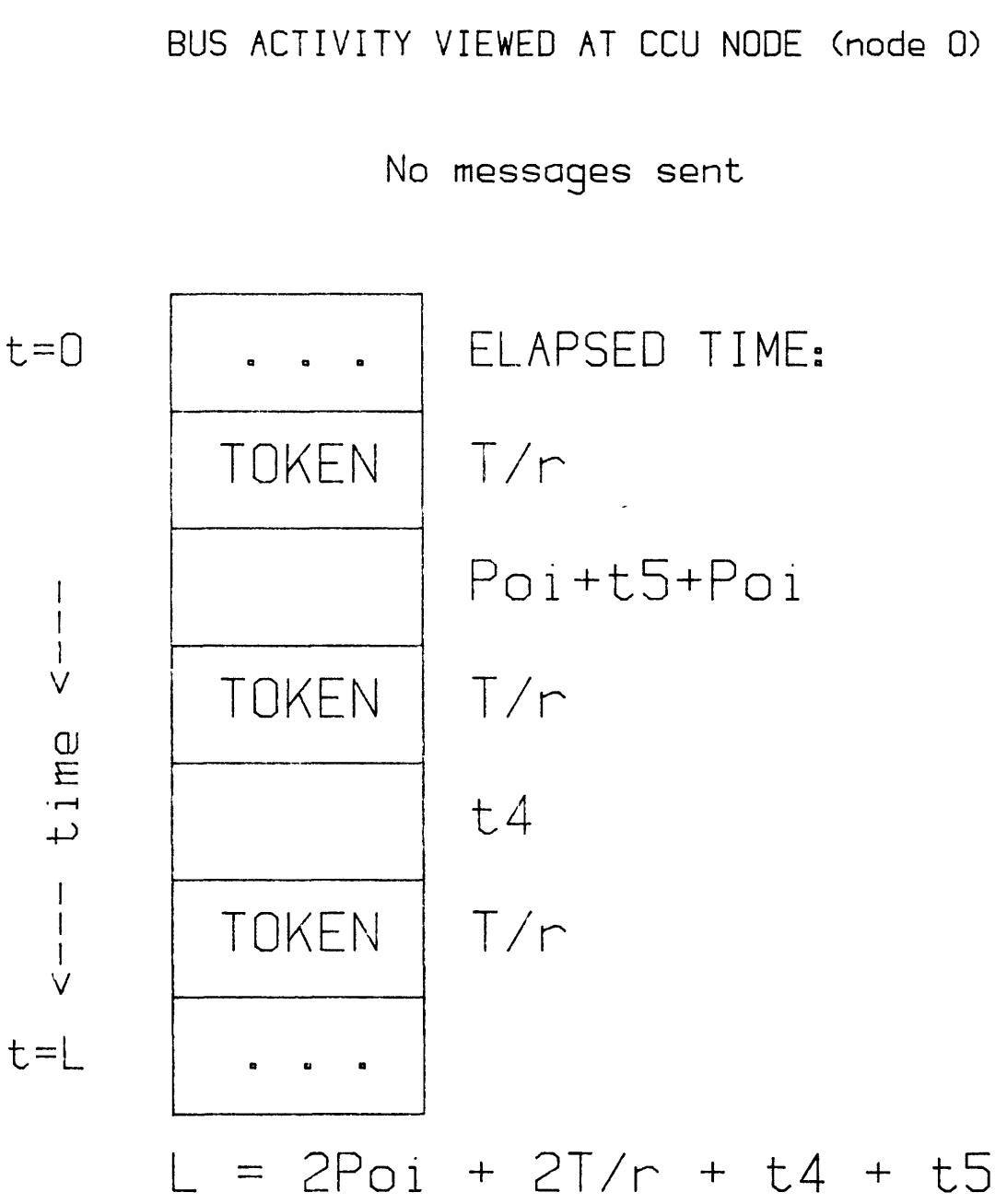


Figure V.2: Bus Activity (no messages)

time, LT,

$$LT = 2\sum_{i=1}^{i=N}(Poi) + 2NT/r + N(t4 + t5) + t6-t4.$$

The propagation time sums in all of the formulas can be simplified by the assumption that all stations are separated by an average distance d. This assumption is particularly valid for large N, since the stations need not be separated by exactly d in each case, but the average interstation separation is d. With this assumption,

$$2\sum_{i=1}^{i=N} Poi = 2\sum_{i=1}^{i=N} id = N(N+1)nd/c.$$

Where c/n is the propagation velocity on the communication medium with c being the speed of light. Therefore,

$$LT = N(N+1)nd/c + 2NT/r + N(t4 + t5) + t6-t4.$$

For ST, the propagation term also includes a Pij term, where j has not been specified. Clearly, there is a best and worst case for that sum, corresponding to transmission to nearest neighbor and to most distant station, respectively. For the best case, Pij is simply d, and thus

$$STb = N(N+1)nd/c + 2Nnd/c + N(2T+A+M)/r + N(t1+t2+t3+t4) + t6-t4$$
$$= N(N+3)nd/c + N(2T+A+M)/r + N(t1+t2+t3+t4) + t6-t4.$$

In the worst case, the farthest distance is the same for any two stations equidistant from the center of the

bus, therefore,

$$2\sum_{i=1}^{i=N} Pij = (N-1 + N-2 + N-3 + \ldots + N/2)\,4nd/c \quad N \text{ even})$$

$$= N(3N-2)nd/2c \quad \text{and}$$

$$STw = 5N(N)nd/2c + N(2T+A+M)/r + N(t1+t2+t3+t4) + t6-t4.$$

However, while the worst case may be valuable as a absolute maximum, neither worst nor best case fit well with the assumption that communication is evenly distributed. For large N, the average propagation time, AP, will be more significant:

$$AP = \sum_{i=1}^{i=N} \{[1/(N-1)] \sum_{j=1,j\#i}^{j=N} Pij\}$$

$$= [2nd/c(N-1)] \sum_{i=1}^{i=N} (1 + 2 + 3 + \ldots + i-1)$$

$$= N(N+1)nd/3c \text{ and thus}$$

$$AST= 4N(N+1)nd/3c +N(2T + A + M)/r +N(t1+t2+t3+t4) + t6-t4.$$

C. Theoretical Values of Performance Parameters

From the definitions of part A and the model of section B, the theoretical values of the performance parameters follow quite simply. The access delay, characterized by system loading (i.e. length and number of messages sent) is given by the maximum loop time, ST, so that the maximum

access delay, MAX AD, is given by:

$$MAX\ AD = 4N(N+1)nd/3c + N(2T+A+M)/r + N(t1+t2+t3+t4) + t6-t4.$$

M is understood to correspond to the average message length, and the maximum value of access delay is the result of M being equal to the maximum allowable message length. In the special case when M is zero, the so-called minimum access delay is the total loop time LT, so that

$$MIN\ AD = N(N+1)nd/c + 2NT/r + N(t4 + t5) + t6-t4.$$

A more detailed definition of M is needed to determine the efficiency. The message frame includes non-useful information, thus M is expressed as m+O where m is the useful information and O is the non-useful overhead. With this definition, the efficiency is the ratio of Nm/r to the total maximum loop time:

$$E = (Nm/r)/[4N(N+1)nd/3c + N(2T+A+M)/r + N(t1+t2+t3+t4) + t6-t4]$$

$$= m/[4(N+1)rnd/3c + (2T+A+M) + r(t1+t2+t3+t4) + r(t6-t4)/N].$$

The throughput is the product of the collective input data rate, R, and the efficiency. The upper bound on throughput is for R=r, since data cannot leave the network any faster than the bus data rate. In this case, the throughput is given by:

$$TP = rm/[4(N+1)rnd/3c + (2T+A+M) + r(t1+t2+t3+t4) + r(t6-t4)/N]$$

$$= m/[4(N+1)nd/3c + (2T+A+M)/r + (t1+t2+t3+t4) + (t6-t4)/N].$$

There are several qualitative observations which can be made about the calculations at this point. In

configurations where the propagation time contribution is insignificant (as is the case for d << 1 Km), the access delay is linear in N and the efficiency is independent of N. As the data rate increases, the access time decreases and will go to zero for small processing times. However, for the efficiency, the negative effect of the process times becomes larger in direct proportion to the data rate. Thus a highly efficient system may be realized by minimizing processing delays while maximizing the data rate for any value of N.

A final step in the analysis is to determine values for each of the variables used in the model. The most revealing values are those of the processing times t1 through t6 and the overhead variables T, A, and O, since the other variables are specified by the network configuration.

As may be seen in the detailed flowcharts of section IV, the interrupt structure of the BIU and the CCU requires considerable decision-making and peripheral commands in the time from message or token reception to appropriate reaction. Thus, the processing times are a very significant factor in the evaluation of the performance parameters. The actual values of the processing times are obtained from the assembly language program for the BIU and CCU. Each sequence of events associated with a processing time consists of many assembly language commands, each of which in turn requires a given number of clock cycles. The sum of the cycles multiplied by the clock period yields the value of the processing time. These values are generally constant, although the independence of the bus and user interfaces could result in further process delays in the case of both interfaces requiring memory access. However, since the time between successive interrupts is much larger than the time required for memory transfers and

Table V.2 Calculation of Processing Times

1 clock cycle = 2/(6.144 MHz) = 326 nanoseconds

tk	number of cycles	time (milliseconds)
t1	1775	.578
t2	1089	.354
t3	1731	.563
t4	1384	.451
t5	1729	.563
t6	1414	.460

decision-making, the processing times may be safely approximated as constant. Table V.2 shows the calculation of the processing times.

Values of the overhead variables come from the SDLC frame formats and an additional consideration of the network design. The Manchester encoding chip used in the bus transceiver requires 11 bit periods for synchronization at the beginning of each message. Using the definition previously stated that only user input is useful information, the values of A and T are the lengths of the acknowledge and token frames, respectively, plus the 11 bits of synchronization each. O is the number of bits in a message frame less the length of the actual message plus the 11 bits. The calculated values of all the model variables are shown in table V.3.

Table V.3 Values of Variables for Prototype Network

Variable	Meaning	Value
T	token frame length	59 bits
A	acknowledge frame length	67 bits
O	message overhead	67 bits
m	message length	8 to 2040 bits
c	speed of light	300000 km/sec
	--design parameters--	
r	bus data rate	38.4K bits/sec
N	number of stations	3
n	refractive index	1.47
d	interstation distance	100 meters
	--performance parameters--	
MAX AD	maximum access delay	180 milliseconds
MIN AD	minimum access delay	12 milliseconds
E	efficiency	88.6%
TP	throughput (maximum)	34K bits/sec

Section VI. Simulation of Data Traffic

The theoretical values of the performance parameters were verified by actual operation of the prototype network. To facilitate this verification, a data traffic simulator was constructed. The simulator was required to generate data, move it through the network, and provide timing and accuracy checks. Part A of this section discusses the design objectives for the traffic simulator, part B is devoted to a description of the operation of the actual system, and part C reports the results of traffic simulation.

A. Simulator Design Objectives

The traffic simulator essentially performs the inverse operation of the prototype network. The prototype network takes data from many sources, channels this data through a single data path, and transfers it to different destinations. The simulator must generate data, send it to many stations, and then retrieve it from many stations. Combined with the objective of measuring performance parameters, this function suggests several of the fundamental design requirements:

1) The simulator must be capable of effective data rates approaching that of the network bus, at least on the short run.

2) The simulator must have capability for several input/output channels.

3) The simulator should generate data either from a random process or some kind of programming.

4) In order to implement error-checking, the simulator must have storage space for at least one message.

5) The simulator must be able to time I/O operations accurately.

In addition to the above considerations, another practical goal was considered in the design process. While

the traffic simulator was to be a valuable diagnostic tool, it was not to be used as part of a final operational system. Thus, an effort was made to produce as little highly specialized hardware as possible, endeavoring to use more general purpose equipment and software. This goal mitigated against the construction of a distributed system, such as creation of a data source for each station with some kind of central control, in favor of a central programmable source, such as a small computer, with multichannel I/O capability. By making the simulator software intensive, its flexibility and future usefulness were greatly increased.

The need for high data rates suggested the use of parallel formatting (e.g. data bytes derived from the logical sense of eight separate wires) as opposed to the serial format (bytes derived from eight or more consecutive pulses) used by both the original user interface and the network medium. The associated trade-off is a decrease in cable length and greater cabling expense, but since the simulator was to be an diagnostic tool only, these problems were not considered troublesome.

Design goals 3, 4, and 5 were easily satisfied by any of a group of desktop computers, but few of these computers offered multiple I/O channels. This problem was solved by using a popular parallel interface, the IEEE-488 1978 standard interface [16]. Briefly, the IEEE-488 interface (originally marketed as HP-IB by Hewlett Packard) is a bus structure using a polling algorithm. Using a pull-up bus structure, many devices maintain constant connection to 16-line bus, eight lines for data and eight lines for control and handshaking. When a given device is active, it manipulates the appropriate lines by grounding them, while inactive devices merely allow their lines to float high. A single unit is designated as system controller. All other

units may send or receive data only by command of the system controller. The system controller initiates and monitors all transactions and may send or receive data as well. Because the system transfers data by handshake (interchange of control lines), the data rate is limited only by the speed at which active devices can execute the handshake. Typically this translates to a data rate as high as 25K bytes per second or higher. Two desktop computers, a Hewlett-Packard 9825 and a Hewlett-Packard 86, were available for use as IEEE-488 controllers. Because of the higher speed capability of the HP-9825 and the greater programming flexibility of the HP-86, both were used in different parts of the simulation.

As was mentioned earlier, however, the prototype network was designed for a serial interface (RS-232) at the user level. Thus, the major work in creating the simulator was developing the IEEE-488 user interface. This work was not wasted, however, because of the wide use of IEEE-488. Each of three prior demonstrations of the prototype network had used at least two IEEE-488 compatible data sources. Moreover, the hardware design of the prototype BIU's had anticipated such possibilities by concentrating the user interface in the 8741A. To accomplish the IEEE-488 interface, the only hardware changes was the replacement of the 8251 USART with pull-up line drivers (Texas Instruments SN75160 and SN75161). All other changes were accomplished by a new program for the 8741A. Appendix A describes the actual programming of the IEEE-488 user interface in detail.

B. Operation of the Traffic Simulator

Before the simulator was actually programmed and used, two helpful troubleshooting tools were constructed. One was simply an LED monitor of the state of the IEEE-488 data and control lines, the other was a slow handshake device,

which was actually a user interface without an associated BIU programmed to add about a second of delay to each data handshake. The slow handshake device, when activated, slowed the IEEE-488 data rate to a speed at which the byte-by-byte transfers could be verified visually. These two devices, as well as a parallel logic analyzer were not used in the actual operation of the simulator, when speed was an absolute necessity, but they proved invaluable to debugging and confidence testing of the simulator.

The simulator itself consisted of only a desktop computer connected to the IEEE-488 user interfaces of the BIU's, thus the heart of the system was software. The software was designed to take full advantage of the features of the interface while exercising and analyzing the prototype network. Before any tests were conducted, the network bus data rate was measured with a frequency counter and the value recorded. This value was continually updated to account for any temperature effects. Three basic tests were designed, a no-message test, a steady-state medium message test, and a short-term maximum message length test.

The no-message test required no simulator action at all. For this test, the network was allowed to operate with no user input at all, which meant that each station sent only the token message when the token COUNT matched MYADDR. The transmitter enable line of one BIU was monitored with an oscilloscope, yielding a periodic signal. In this mode, the period of the signal corresponded to the minimum access delay. The test was conducted for one, two, and three-station network configurations.

The steady-state test required use of the simulator. In this test, the goal was to provide as even a flow of information through the network as possible. In this test, a random message (created by use of a psuedo-random number generator and decimal to ASCII conversion) was

routed to a given station to be sent to some other station. As soon as the transfer to the sending station was completed, a timer was started. The destination station was monitored until the message was successfully recovered, at which time, the timer was stopped and the message checked against the original message for any errors. The elapsed time was adjusted to take account of any delays inherent in the simulator, determined at the beginning of the test. The value of the message length divided by the network bus data rate was calculated. This value divided by the elapsed time corresponded to the efficiency. The length of the message divided by the elapsed time corresponded to the throughput. This test was conducted on a long-term basis, several hours at a time until the measured parameters became constant to at least three places.

The third test was essentially like the second, except that it involved a series of tests instead of a single continuous test. The goal of this test was to measure performance of the network at full loading. Unfortunately, this test could not be performed continuously for all data rates because of a limitation in the simulator data rate. The user interface was found to be able to operate IEEE-488 at a rate of about 12K bits per second at best, the limitation being the user interface's handshaking ability. Thus, the simulator could only maintain a 12K bits per second effective data rate. However, on the short run, that data rate could be multiplied by the number of stations by taking advantage of the multiple listeners capability of IEEE-488, which allows any number of listeners (data receiving stations), but only one talker (transmitter). In this test, all stations were addressed to listen as a basic message was sent, without the closing character and address. Then each station was given an individual closing character and destination address in quick succession. The effect is of N nearly simultaneous messages being sent.

A timer was started and is not stopped until each station indicated it had a message ready for the user. The messages are then read back by the computer and checked for errors. The elapsed time corresponded to the maximum access delay. The sum of the lengths of the messages divided by the elapsed time corresponded to the throughput. The throughput divided by the data rate corresponded to the efficiency. Because of the margin for error in this test, the average was not computed immediately, but only after several iterations and the discarding of clearly erroneous values.

C. Experimental Data

In general, the experimental data indicated that the model of Section V. was accurate, although the narrow range of the network bus data rates (up to 64K) and number of stations (up to 3) could not supply far-reaching results. The model did seem to underestimate the magnitude of the delays by a small margin, which became more significant at higher network data rates and longer message lengths. This would seem to indicate additional delays due to memory transfers, ignored in the model, were somewhat of a factor. At no time did the experimental values fall outside of 15% of the predicted values.

Using the first test, the minimum access delays were found to be essentially linear in N, as expected, corresponding to insignificant propagation delays. The test was conducted for three values of N and four values of r. As a function of r, the delay was found to be a sum of fixed delay and a linear term in r. The least squares coefficients of fixed and r-dependent delay were found to be

$$\text{MIN AD} = N(123r + 1.24 \text{ msec}).$$

These calculations displayed a correlation coefficient value of 0.85. The projected value of the coefficient for r was 118, and the projected value of the fixed delay was 1.15 msec. These results seem to indicate an additional amount of both fixed and variable delay, but justify the approximations made in Section V.

The second test showed efficiency to be essentially independent of N. This test was conducted for three values of N, five values of m, and four values of r. Assuming N independence (the least-squares coefficient of N was found to be less than .001), the equation for efficiency was

$$E = m/(262 + m + .0021r)$$

and the equation for maximum access delay was found to be

$$MAX\ AD = N(262 + m + .0021r)$$

For these values, the correlation coefficient was determined to be 0.81. The value 262 corresponded to the fixed overhead, projected at 252, while the coefficient of r corresponded to the sum of processing times, projected at 1.95 milliseconds. As before, both terms were slightly higher, indicating additional delay not accounted for in the model, perhaps as a result of memory transfers and transceiver delays. However, since the average of the observed data fell within 10% of the model, the model was considered to be sufficiently accurate.

The third test proved less consistent than the second, due to the indeterminant nature of token count at the start of each test. While the same general form was found for the efficiency and delay equations, both delays averaged considerably higher:

$$E = m/(408 + m + .0035r)$$

$$MAX\ AD = N(408 + m + .0035r)$$

The correlation coefficient here was only 0.48, giving another indication that the test was not totally accurate. These results indicated that the test was not achieving simultaneous completion of the data messages, with at least one station missing an opportunity to transmit because its message was incomplete. Thus, only the form of the results of this test was considered valid, with the second test viewed as the more accurate determination of the values of the coefficients. The results of the third test were viewed as more convincing proof of N-independence of the efficiency (the coefficient of N was on the order of .001).

In both the second and third tests, error-checking was conducted. Because of the error-checking performed by the network itself, two kinds of errors were recorded. If an error was detected by the network, the entire message frame was discarded and not forwarded to the user (the simulator). This was termed a detected frame error. If an error was detected by the simulator but not the network, an undetected error was said to exist. In processing over 10 million frames, no undetected errors were found and less than 100 detected errors were indicated. Because a frame consisted of on the order of 100 bits, the detected bit error rate of the network was less than one in 10 million, and the undetected error rate still less.

In summary, then, the simulation of data traffic validated the conclusions of the performance model, although indicating slightly additional delay. The network was shown to be able to maintain steady, high-volume usage without instability. In a generally safe environment, errors were few, and the errors which did occur were detected by the network. Thus the prototype network satisfied the majority of its original design objectives.

Section VII. Network Revisions

Based on the results of the analysis and simulation, alterations to the original prototype network were proposed. These changes included efforts to improve network reliability and improve performance. Part A of this section describes reliability improvements, part B discusses performance improvements, and part C recalculates the performance model based on the changes made.

A. Reliability Improvements

While Section VI. detailed the data accuracy of the prototype network, no mention was made of component failures; the performance model of Section V. was founded on the assumption that all components operated correctly and to specification. Particularly during the early development of the prototype network, operation was severely affected by inability of the Intel 8273 to function correctly. The failure was premature and unnotified disabling of the receiver, caused by internal transitions on the modem control lines. As a result, many messages were lost and the network had to be periodically (every few seconds) reset. Before the simulation of traffic was begun, this problem required at least a temporary solution. After much experimentation, the problem was localized to the modem control lines (RTS and CTS). By changing the gating of these control lines, stability was increased to the order of minutes before requiring reset, and finally, by tying CTS low at all times, the problem was essentially eliminated. However, this problem combined with the 8273's data rate limit of 64K, was seen as a clear indication that the 8273 should be replaced.

The selection of a replacement SDLC protocol chip involved the evaluation of many alternatives (Texas Instruments' TMS9903, Standard Microsystems' CDM5025,

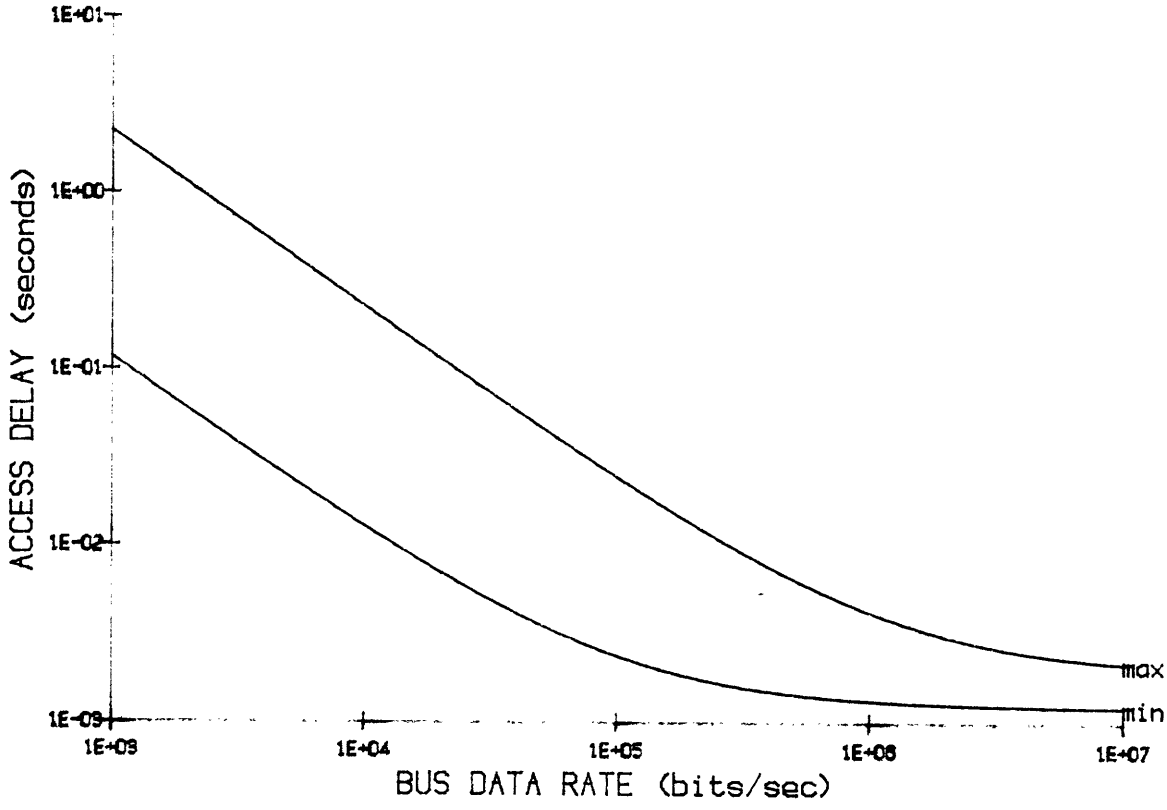
Signetics' 2652, Fairchild's F6856, Western Digital's SD1933, NEC's uPD7201, Intel's 8274 and others) [17]. The goals for this chip were data rate on the order of 1 Megabit/sec, DMA capability, SDLC formatting, dual receive addresses, and modem control lines. Compatibility with the existing network was seen as required. With these goals and requirements, the choice became essentially limited to the Intel 8274 and the NEC uPD7201. Due partly to the bad experience with the 8273, and after successful laboratory evaluation of the uPD7201, the uPD7201 was chosen, despite its different implementation of DMA control lines.

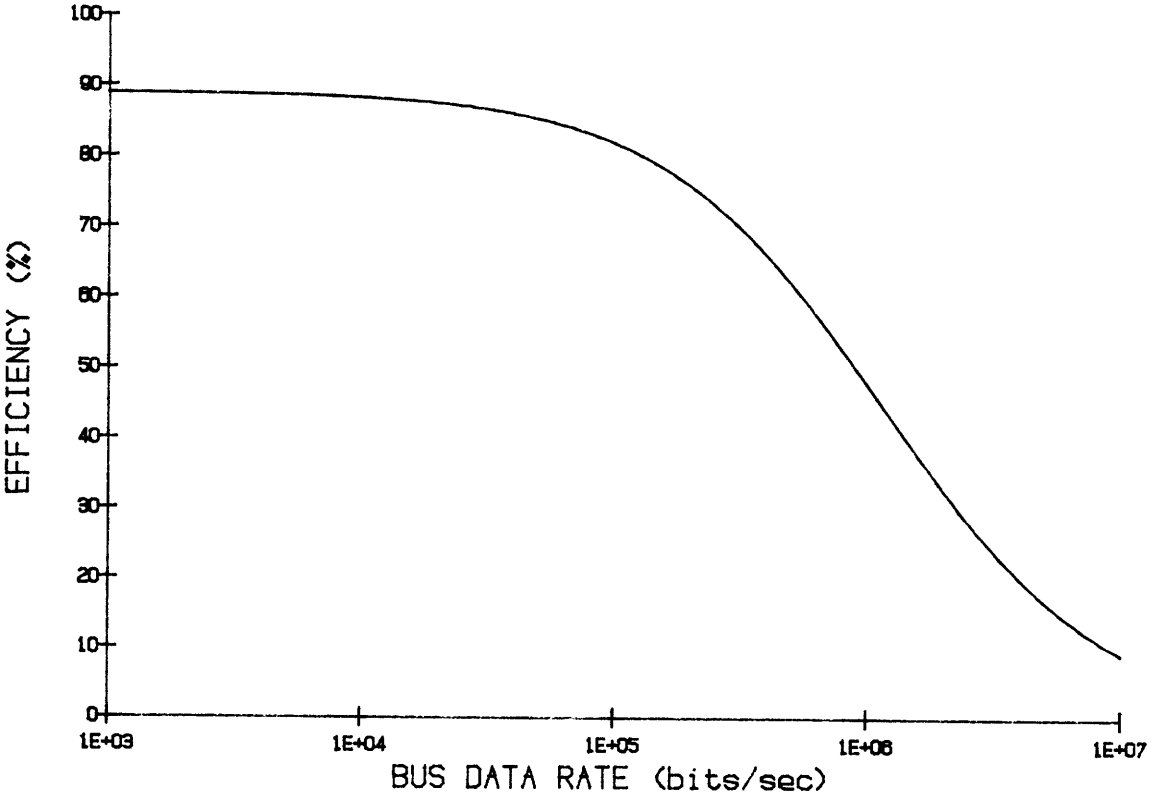
The appealing features of the uPD7201 were its data rate to 880K, dual independent SDLC channels, and simplified command set. The difference in DMA operation involved the use of a single DMA acknowledge line for four channels, but only simple gating was required to interface with the 8257 DMA controller. With this change in the hardware and software designs, there were also several other changes proposed to improve performance.

B. Performance Improvements

One of the first improvements to be made in the prototype network was to be the addition of several more stations, since the three station network would not be adequate for most applications. Because the access delay is a linear function of N, however, an increased number of stations quickly leads to maximum delays on the order of seconds, which is generally unacceptable. To minimize these delays, the data rate r must be increased and the processing times minimized. The effect of increasing the data rate is to decrease efficiency and throughput; this effect becomes dramatic in the region where r nears 1M bits/sec. The cause of the sharp decrease in efficiency at high data rates is the processing times; thus, any attempt

Figure VII.1
Access Delay and Efficiency vs. Data Rate





to increase the data rate (decrease access delays) must be accompanied by reductions in the processing times to maintain high efficiency. The access delay per station and the efficiency (roughly independent of N for short distance configurations) are shown as a function of the network data rate in Figure VII.1.

To minimize the processing delays, four changes were made: a revision of the acknowledgement scheme, a redefinition of the token message, restructuring of the BIU software, and a change in the bus transmitter control line. From the detailed flowcharts, it can be seen that the majority of the processing time is spent setting modes or issuing commands to peripherals, or waiting for a response. The goal of the changes is to minimize this time by eliminating all waiting and performing set-up commands before the token count reaches the value.

In the prototype network, after a station sent a message, it waited for an acknowledgement. This added two processing times and twice the interstation propagation time to the delay. For the revised network, a different acknowledgement scheme is employed. When a station receives a message from another station, it makes no immediate response, but records the fact that it needs to acknowledge the message. Then, when its next transmission turn arrives, it first transmits any acknowledgements, followed by a message if it has one. Stations are still allowed only one message per turn, but may add any number of acknowledgements. A sending station knows that its message was not received correctly if no acknowledgement has been received by the time its next turn arrives. At this time, the station may re-transmit its message.

The token message is the most common message on the network. However, since there is no other information involved in the token other than its existence, there is no

need to frame the token message as an SDLC frame. For the revised network, a single sequence (10 bits) is proposed as the new token message. This sequence is chosen as ten consecutive ones, which corresponds to the SDLC abort code. As was stated in Section III, the SDLC format never allows more than five consecutive ones in the data, using six ones for the flag. When an SDLC receiver receives more than six consecutive ones, it aborts the current data stream being received and signals reception of the abort code. This method works perfectly for the token. When the receiver interrupts the processor, the processor must check its status to determine the cause of the interrupt. If the receiver indicates that an abort was received, the processor merely increments the value of COUNT and continues operation. The uPD7201 has a special command for sending the abort sequence.

Beyond the savings of overhead from redefinition of the token, the abort sequence allows for the elimination of another processing time. For the revised network, the processor monitors COUNT as before. However, when the token count is one less than the station's address, it issues the command to its transmitter to send a message. The transmitter is not able to transmit immediately, however, due to a special circuit used to provide the CTS signal. This circuit is essentially a shift register used to monitor the last eight bits received from the network. As soon as eight consecutive ones are received, the circuit allows CTS to go low, and transmission follows reception of the token by no more than a gate delay. At conclusion of the transmission (messages and/or acknowledgements followed by the abort code), the transmitter relinquishes RTS and the token recognition circuit is disarmed until the processor issues another transmit command. The heart of this operation is the dual independent channel capability

of the uPD7201. Channel A is used for receiving only and is always active. Channel B is used for transmitting only, and is only active after processor initiation and the next token reception.

A final revision concerns the user interface. Because the central processor may keep the 8741 on hold longer under the new acknowledgement scheme, provision is made for more efficient storage by the 8741. In the prototype network, the user did not forward the destination address until the conclusion of the message. For the revised network, the first two characters received after the initiating character are interpreted as the destination address. This structure has two main advantages: the 8741 may perform any necessary division of the message into blocks for immediate transmission, and the local buffer of the 8741 may be used more fully. In the prototype network, if a message was started but not completed, the 8741 could not send data in either direction until a timeout expired. Under this new structure, the 8741 may save blocks of data up to 32 bytes, and transfer them to memory at once. If the user lags in input, the partial message may be sent over the network to be followed by a conclusion. While on hold, the 8741 may continue to receive data, up to 32 bytes before it must hold off the user. Although this change does not directly affect the performance model, it can be seen that it will effectively reduce access delay.

C. Recalculation of the Performance Model

Using the same variables and procedure as the prototype analysis, the benefits of the network revisions may be demonstrated. In the case that no messages are being sent, the sequence of events is the same as it was for the prototype network. For each station the bus, the minimum loop time, L, is given by

$$L = 2Poi + 2T/r + t4 + t5.$$

However, the magnitudes of t4 and t5 have been reduced from milliseconds to microseconds and the magnitude of T has been reduced by a factor of five. As before the minimun access time becomes

$$MIN\ AD = N(N+1)nd/c + 2NT/r + N(t4 + t5) + t6-t4.$$

In the case when each station sends a message, the sequence of events is different due to the elimination of the wait for acknowledgement. In this case, as shown in Figure VII.2, the activity observed at node 0 commences with the token as before. After the token has passed the node, there will be a wait of 2Poi as the token travels to node i and a message comes back from node i. There also is a waiting time t1, corresponding to the token-to-message delay, which is greatly reduced in this network. For steady state operation, there must be one acknowledgement for each message sent. Thus, an acknowledge frame is assumed to be the first part of the message, passing the node for a time A/r. This is followed by a message taking M/r, and finally the token, T/r. After the token passes, the CCU has a delay t4 before the token is sent again, completing the cycle. Thus the maximum loop time S is

$$S = 2Poi + (2T + A + M)/r + t1 + t4.$$

Again, the values of T, t1, and t4 are greatly reduced, and the 2Pij term vanishes along with t2 and t3. The maximum access delay becomes

$$MAX\ AD = N(N+1)nd/c + N(2T + A + M)/r + N(t1 + t4) + t6-t4.$$

And finally, as before, the efficiency and throughput

BUS ACTIVITY VIEWED AT CCU NODE (node 0)

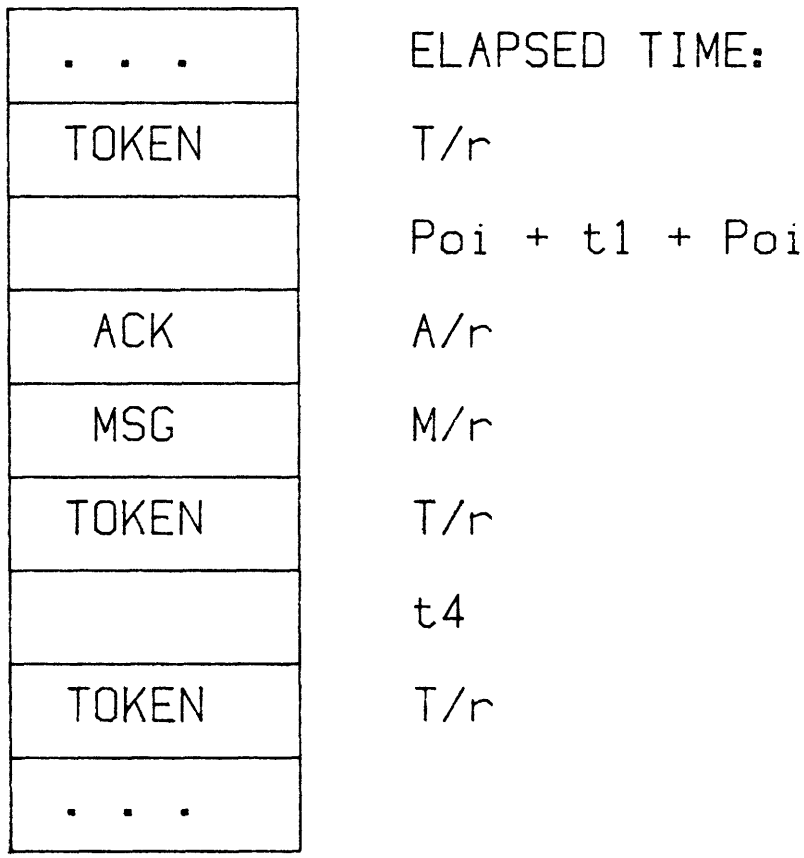


Figure VII.2 Revised Bus Activity (with messages)

$$S = 2Poi + (2T + A + M)/r + t1 + t4$$

are given by

$$E = m/[\ N(N+1)rnd/c + (2T + A + M) + r(t1 + t4) + r(t6-t4)/N],$$

and $TP = rE$, respectively.

While the formulas may not seem to be much different from those of the prototype network, the differences are in fact quite significant. To this end, Table VII.1 and Figure VII.3 show the improved performance for comparison to Tables V.1 and V.2 and Figure VI.1.

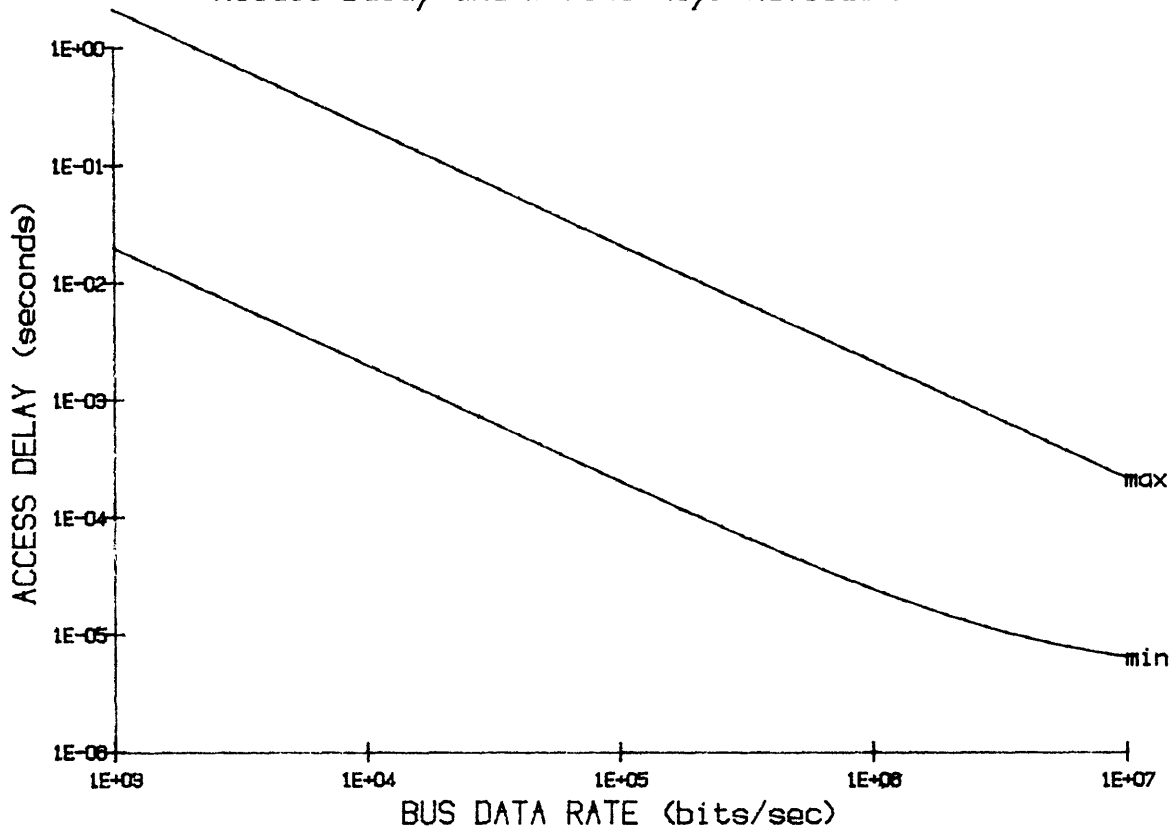
Again some qualitative observations are important. With the processing time effectively made insignificant, the propagation times are no longer totally insignificant. Thus the system does not achieve access times in the nanoseconds when d is on the order of hundreds of meters. However, both the propagation times and the processing times do not make an overwhelming contribution to the delays until r exceeds 1 Megabit/sec or N exceeds 20. Since both of these cases are on the extreme limits of the intended operation of the network, the revisions are most useful.

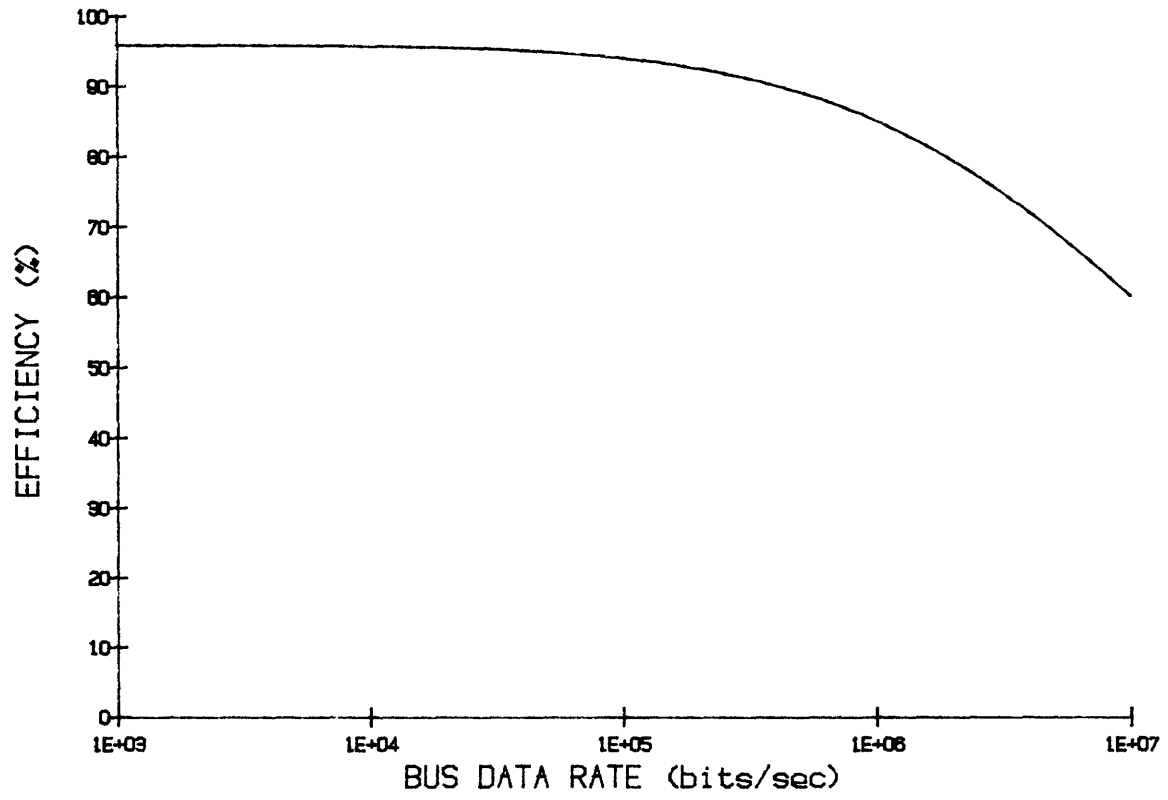
Table VII.1 Values of Variables for Revised Network

Variable	Meaning	Value
T	token frame length	10 bits
A	acknowledge frame length	67 bits
O	message overhead	67 bits
m	message length	8 to 2040 bits
c	speed of light	300000 km/sec
t1	processing time (BIU)	500 nanoseconds
t2	processing time (CCU)	500 nanoseconds
	--design parameters--	
r	bus data rate	880K bits/sec
N	number of stations	up to 32
n	refractive index	1.47
d	interstation distance	100 meters
	--performance parameters--	
MAX AD	maximum access delay	
	3 stations	7.68 milliseconds
	32 stations	80.5 milliseconds
MIN AD	minimum access delay	
	3 stations	75.7 microseconds
	32 stations	1.23 milliseconds
E	efficiency	
	3 stations	93%
	32 stations	95%
TP	throughput (maximum)	
	3 stations	818K bits/sec
	32 stations	838K bits/sec

Figure VII.3

Access Delay and Efficiency: Revised Network





Section VIII. Conclusions

At the time of this report, the revised network is still in the development stages. After a working model is completed, the traffic simulator will again be used to verify performance before a final circuit board version is produced. Timing will be much more critical in the 880K version because the network data rate is approaching the clock rate of the microprocessors. If necessary, small delays may be inserted to increase the magnitude of t1 and t4 to insure correct operation. However, these delays will not have the effect of the processing delays of the prototype network due to the associated software redesign. An alternative for networks of many stations would be to arm the transmitter two or more tokens in advance of the desired count. Regardless, the prototype network and the revised design demonstrate that a network of this sort may achieve high efficiency and acceptable access delays, despite heavy loading and many users.

The value of the broadcast token protocol can be seen by comparison to the chief competitor, Carrier Sense Multiple Access protocols, such as that used in Ethernet and other popular local area networks [18]. Because of the nature of the protocol, CSMA networks may experience instability or seriously degraded performance under heavy loading. Generally, such networks operate at efficiency (utilization) levels less than 20% in order to assure proper operation. Thus the revised network may achieve up to five times the data throughput using the same network data rate; moreover, there are no problems with instability. With appropriate buffer space, the revised network could be designed to maintain total transparency at constant effective data rates up to that of the network bus, even exceeding that rate on the short run.

The final parts cost for the network is under $150 per

station for the wire pair bus, with fiber optic or hybrid stations costing less than $800 per station (Appendix B describes the cost differential for various implementations of the network bus). Thus, while the added cost of the optical fiber may not be justified for all locations, where needed it is available at a reasonable price. All parts are commercially available and most are readily second-sourced.

As designed, the network is not intended for high-volume long-length transfers, as in computer-to-secondary memory applications. However, the network is ideally suited for high-volume short transfers, such as process control, factory communications, and small scale computer interfacing. Such a network would be well suited to local data handling such as electronic mail or interconnection of word-processing equipment.

Thus the network seeks the middle ground, providing a flexible interface to a wide variety of equipment. Although not capable of the highest data rates in local area networks, it does not come with the highest price either. The available data rate is used with a high level of efficiency, and operation is highly reliable. Combining flexibility, simplicity, and low price, the network is well suited to most small and medium scale applications.

Appendix A. An Approach to Assembly Language Progamming

While the actual assembly language programs used for the CCU and the BIU are not particularly noteworthy, the process used for converting the detailed flowcharts of Section III to assembly language is significant. This appendix details this process and uses sections of the code for the IEEE-488 (hereafter, HP-IB) interface as an example.

Ideally, the programmer should have little need for assembly language programming, instead relying on the simplicity of a higher-level languages. However, for systems such as the prototype network where memory and program storage are limited, the structure of a higher-level language imposes a prohibitive space requirement. Thus assembly language must still be used to tailor software to the small system. While the unflagging consistency of a compiler/interpreter leads to waste on the small scale, this consistency must be emulated by the assembly language programmer if reliable and readable code is to be generated.

The major focus of an assembly language program is organization. The most critical design objectives are often not the actual software, but are contained in the assignment of memory and I/O, and in the stucturing of the program itself. For most small-scale systems such as the prototype network, the majority of the function is I/O related; thus careful design of the I/O portion of the system is essential for short, efficient programs.

The goal of the process used for the assembly language programming of the prototype network is to show that with proper preparation and organization, the commonly accepted standard of 10 lines of documented code per man-day can be easily exceeded. This is accomplished by allowing the

programmer to act as a highly intelligent compiler, working within the context of a strict organization, but able to handle special situations with flexibility. This process is described by the five steps listed below.

1. Preparation: The programmer must be fully acquainted with the processor to be used, its assembly language and the associated assembler. It is better to write a few sample programs before starting the actual programming than to write code which evolves as it goes. After the programmer feels comfortable with the language, he should make a second reading of all appropriate data books for the express purpose of compiling a list of peculiarities to be observed in programming. These peculiarities often are the cause of malfunction if ignored; anticipating them prevents trouble.

2. Assignment: Based on the detailed flowcharts and specifications, memory and I/O pins must be assigned. Memory assignment includes register and main memory allocation. As a rule of thumb, no fewer than three registers (at least one of which may be used as a memory pointer for indirection) should be saved as work registers. Remaining registers should be assigned to variables which are used most often or which are used when timing is critical. If flags are needed, one register may be used for up to eight flags using a bit mask (eight bit processors). If the assembler allows, all main memory variables should be described by labels as opposed to absolute locations, in order to promote relocation ability. In a similar manner, I/O pins should be assigned to minimize delay, in careful observance of direction, data sense, and electrical requirements. When this step has been completed, a chart should be made, listing the meaning of all registers, variable labels, and I/O pins. The more detailed this chart, the more straightforward the programming will be.

3. Initial Software: Using the detailed flowchart and the chart prepared from the assignment phase, a first draft of the program is made. If preparation is sufficient, there will be few structural changes in subsequent versions. As the program is written, the trade-off between length of program and speed of operation should be remembered. Use of CALL or JUMP instructions will lessen program length, but increase execution time (for instance, the Intel 8085 requires 19 cycles for a CALL/RET pair [19]). Thus, sections requiring minimum delay should avoid short subroutines. Keys to readable and reliable code are consistency and modularity. In each section, branch labels should be chosen which relate to the function of the section, i.e. START for the first section label. Subsequent labels in the section should be similar to the main section label, as S2 for the second label in section START, or should reveal the process, as WT for a wait loop. When possible, the same sequence of instructions should be used for similar functions; for example, a convention of increment before test in loop structures leads to easier debugging. As already implied, the program should be divided into small sections, each with narrowly defined tasks. Besides providing easier programming, the modular approach often produces sections which may be used in several different programs, or several times within the same program.

4. Debugging: After the initial program has been edited of errors and successfully assembled it may be tested in actual operation. When possible, this testing should be modular as well, testing each function independently. When errors occur, likely trouble spots are omitted lines in the program, incorrect decision testing, or unplanned branching. Each section of the program should be checked to see how it is called, where it branches or returns, and what variables

or registers it affects.

5. Final Program: Any changes or improvements are made in the last step of programming. After these changes are completed, the final program should be documented. While the first draft of the program likely requires few comments, the final program needs extensive comments in order to provide for future use. These comments need not explain every line, but the function of each section as well as any peculiar operations should be carefully explained.

As a demonstration of these steps, the remainder of this section is devoted to an example program. The program is a collection of several sections from the HP-IB interface program for the 8741. Combined in this fashion, it was used as a test for the interface hardware. Sections GETB, LISTEN, OUTB, and TALK are actual sections from the HP-IB interface program, while START was modified specifically for this program. The overall function is to monitor the HP-IB for talk and listen commands addressed to the unit. If addressed to listen, the unit reads data from the bus until a linefeed is encountered. If addressed to talk, the unit sends whatever data it has stored to the HP-IB. Steps 1 through 5 are explained below.

1. HP-IB requires 16 data lines (eight control lines and eight data lines). The 8741 provides 16 I/O lines, but four are dedicated to the DMA and 8085 interface. Thus, additional I/O lines are required. These lines are supplied by the Intel 8243 I/O expander. Texas Instruments SN75160 and SN75161 line drivers are used for driving the HP-IB (see Figure A.1). In addition to the HP-IB signals, the drivers require three inputs: pull-up enable (tied low for HP-IB), direction control (tied high for non-master), and talk enable (high for talk). The expander provides four 4-bit I/O ports, connecting to the lower four bits of the 8741 and the 8741 Sync output. The 8741 provides up to

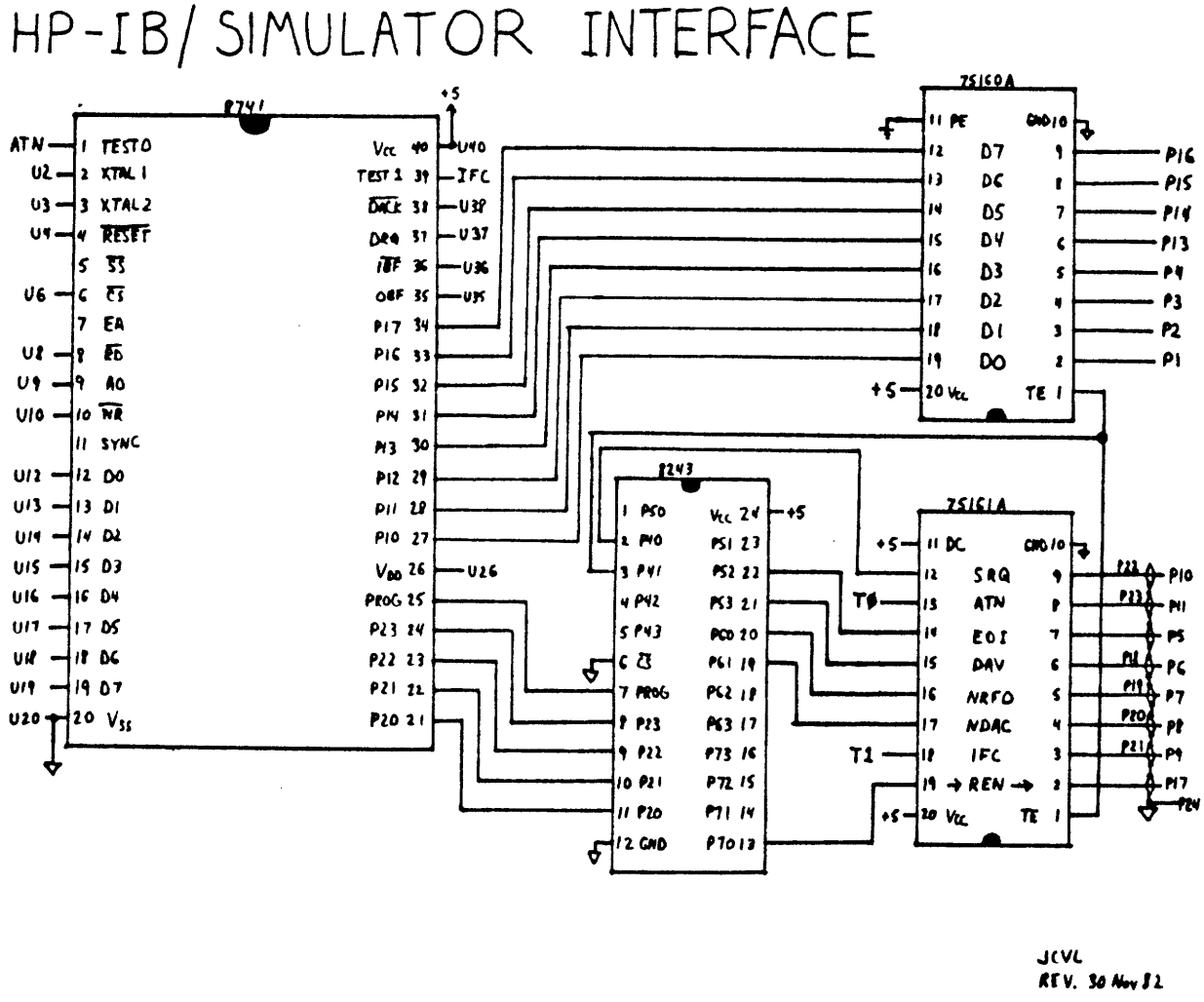


Figure A.1: IEEE-488 Interface Hardware Design

sixteen registers and 48 additional RAM locations. Registers R0 and R1 may be used for indirect addressing of memory. There are 1K bytes of EPROM available for program. In addition, the following peculiarities were noted:

a) Output and input may not be mixed at the same time on any port.

b) A "1" must be written to each input pin of Ports 1 and 2 prior to each reading.

c) For ports 4-7, if the port is changed from output to input, the first reading must be discarded as invalid.

d) An eight level stack occupies locations 8-23 of RAM. If more than eight nested calls occur, the stack overflows into remaining memory.

e)Conditional jumps may not extend beyond page boundaries (four pages of 256 bytes).

2. For this small example, program length restrictions are not a problem. However, because of the repeated use of input and output, a subroutine is written for each procedure. Including the subroutines, the program is divided into five sections, START, GETB, LISTEN, TALK, and OUTB. The assembly-time variable for HP-IB address is designated ADRS. A resulting value, the listen address MYADD is computed from ADRS by the assembler. Reset causes the execution of the statement JUMP 0, therefore the program contains a jump instruction at location 0 to START. Because of the slight need for registers, only one register bank, bank 0, is used in favor of expanded RAM availability. R1 and R2 are designated as work registers, along with the accumulator. R0 is designated as the pointer to memory and R4 is the byte counter. The assignment of I/O ports is more involved, due to the imposed restrictions. Because of their input only nature and their high usage, ATN and IFC are assigned to T0, one of two test inputs. The eight HP-IB data lines are assigned to port 1 to take advantage of

higher-speed transfers and manipulative flexibility. The remaining six HP-IB control lines are assigned to ports 4 through 7 (expander ports) in such a way as to avoid mixing output and input at any time. The final assignment is shown in Table A.1.

3-5. From Table A.1 and the detailed flowcharts (Figure A.2), first and final versions of the program were prepared. Debugging revealed errors in the hardware design corresponding to a transposition of the data lines. After this error and syntactical errors were corrected, an updated version was writtten, documented and assembled (Figure A.3).

Table A.1 Assignment of Memory and I/O

Random Access Memory

Locations	Contents	Function
0	R0	POINTER
1-3	R1-R3	Work registers
4	R4	COUNTER
5-7	R5-R7	Not used
8-23	Stack	
24-63	General	I/O Buffer

Program Memory

0-2	RST INT	Reset interrupt
3-9	INT VEC	Not used
10-1024	General	Program memory

I/O Ports

Port	Pin #	Assignment
1	0-7	HP-IB data lines
2	0-3	Expander port I/O
	4-7	For DMA and 8085 interface
4	0	SRQ (Service Request)
	1	TE (Talk enable)
	2,3	Not used
5	0,1	Not used
	2	EOI (End or Identify)
	3	DAV (Data valid)
6	0	NRFD (Not Ready for Data)
	1	NDAC (Not Data Accepted)
	2,3	Not used

Test Inputs

T0 is ATN (Attention)

T1 is IFC (Interface Clear)

HPTEST: Power-up/Reset

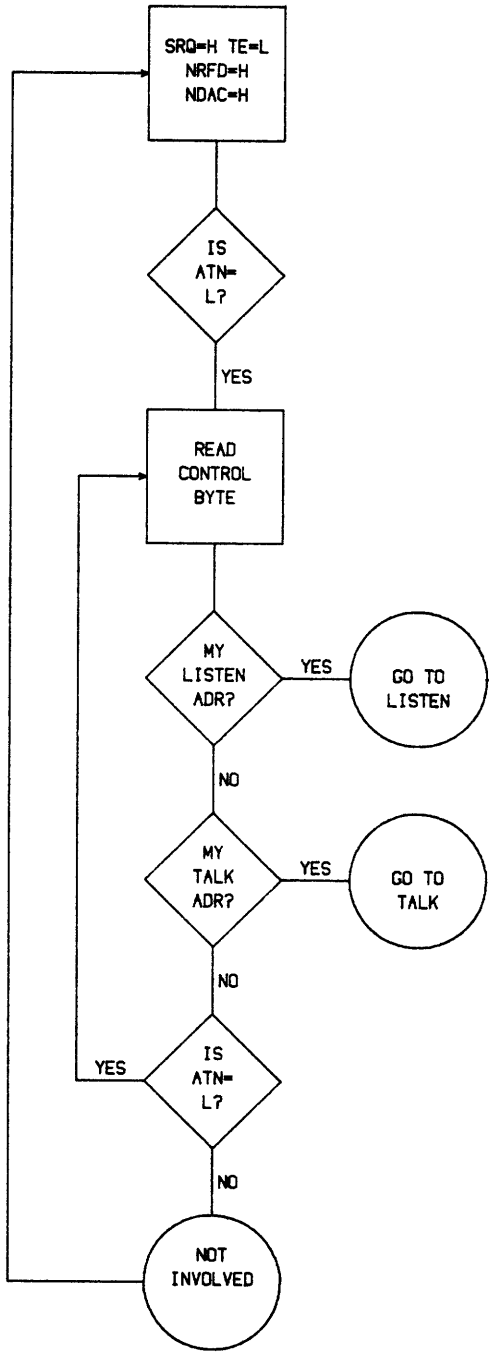


Figure A.2.1 Detailed Flowchart: HPTEST

HPTEST: Read byte (GETB)

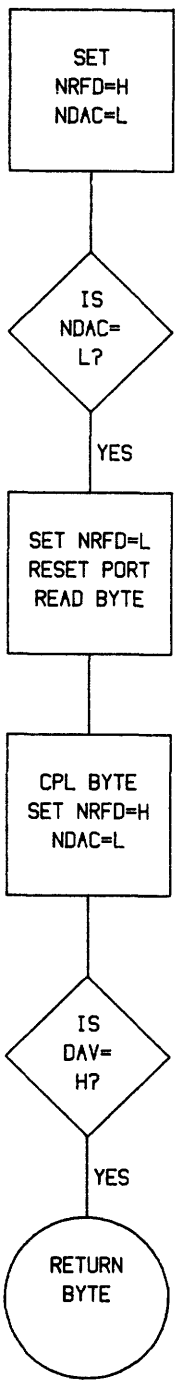


Figure A.2.2 Detailed Flowchart: HPTEST

HPTEST: LISTEN

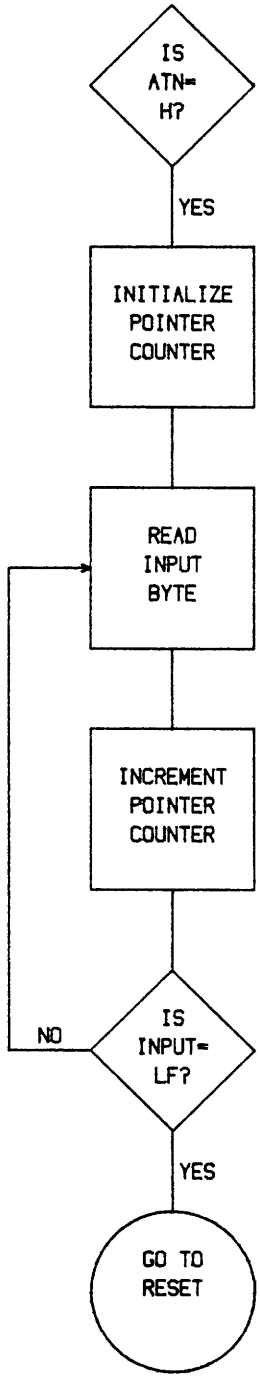


Figure A.2.3 Detailed Flowchart: HPTEST

HPTEST: TALK

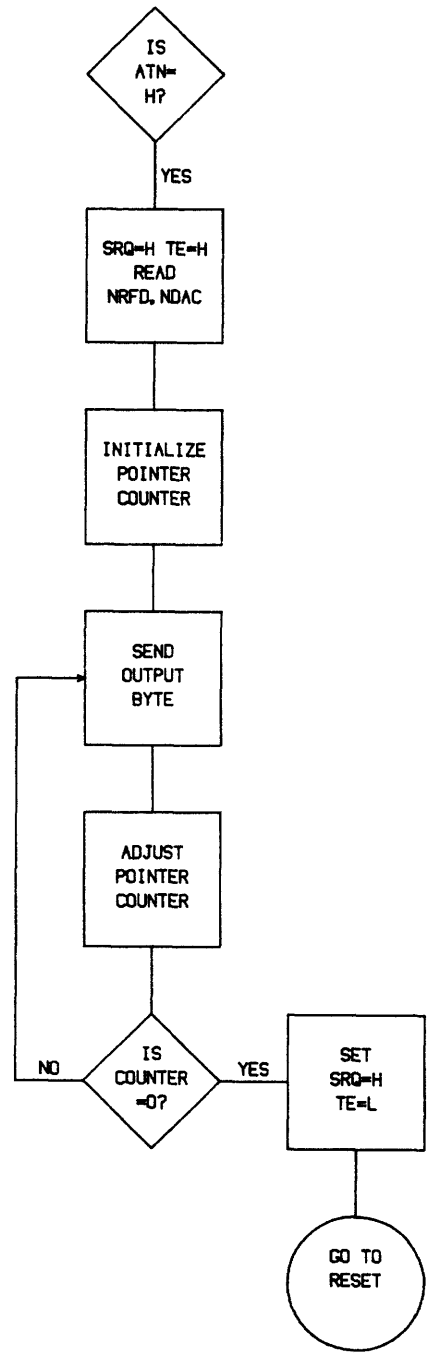


Figure A.2.4 Detailed Flowchart: HPTEST

HPTEST: Send Byte (OUTB)

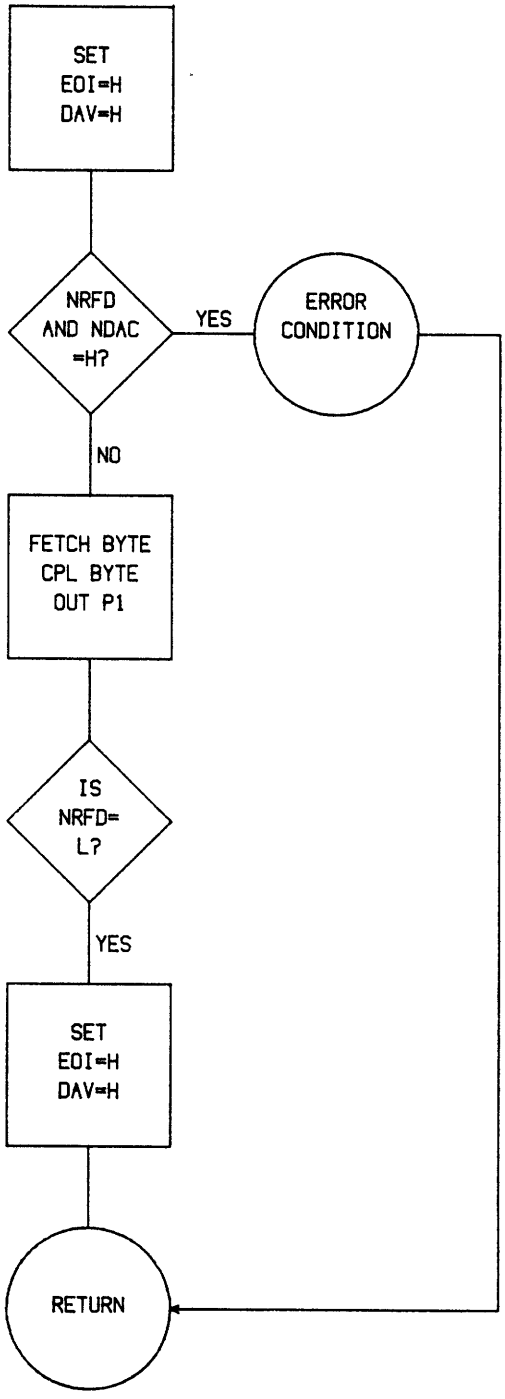


Figure A.2.5 Detailed Flowchart: HPTEST

Figure A.3: Final Program for HPTEST

```
LF      EQU     0AH             ;Constants defined
ADRS    EQU     06D             ;ADRS is HP-IB address (0-32)
MYADD   EQU     ADRS+32D        ;Listen address has bit 4 = 1
ORG 0                           ;Begin actual program
        JMP START               ;Reset forces CALL 0
ORG 10                          ;Skip other interrupt branches
;*************************************************************
; START is the main module of this program.  From power-up,
; HP-IB I/O lines are set for input and control lines set
; high.  Secondary entry point S2 is for resetting control
; lines after reading a control byte or after completion of a
; subprogram.  WT is a waiting point, left only when ATN
; (T0) goes low.  After leaving WT, one control byte is
; read.  This byte is tested against MYADD, the listen
; address.  If there is a match, the subroutine  LISTEN is
; executed.  If not, the residue is tested to see if the
; difference is bit 6 high and bit 5 low (bits 6 and 5 high
; after exclusive or) indicating a match of the the talk
; address, in which case the subprogram TALK is executed.  If
; neither match, ATN is tested again.  ATN still low indicates
; another  control byte is available.  Otherwise, the program
; returns to S2, and waits for ATN to go low again.
;*************************************************************
START:  SEL RB0
        MOV A, #01H             ;Set SRQ high, TE low
        MOVD P4, A
S2:     MOV A, #03H
        MOVD P6, A              ;Set NDAC and NRFD high
WT:     JT0 WT                  ;Wait until ATN is low
LP:     CALL GETB               ;Read a control byte
```

Figure A.3: Final Program for HPTEST-continued

```
        XRL A, #MYADD           ;Match with listen address
        ANL A, #07FH            ;Check only lower 7 bits
        JZ LISTEN               ;Go to LISTEN on match
        XRL A, #060H            ;Match with talk address
        JZ TALK                 ;Go to TALK on match
        JT0 S2                  ;If ATN high, go to S2
        JMP LP                  ;Otherwise, go to LP
;
;
;
;*************************************************************
; GETB is the subroutine used to read the eight data lines.
; It includes all necessary manipulation of control lines DAV,
; NRFD, and NDAC.  At return, the byte read is in the
; accumulator and a back-up copy remains in register R2.
; Only R2 and the accumulator are affected.
;*************************************************************
;
GETB:   MOV A, #01H
        MOVD P6, A              ;Set NRFD high (false)
NDAV:   MOVD A, P5
        ANL A, #08H             ;Test NDAV only
        JNT1 START              ;Reset if IFC is low
        JNZ NDAV                ;Loop if NDAV high
        CLR A
        MOVD P6, A              ;Set NRFD low (reading data)
        ORL P1, #0FFH           ;A 1 must be written to each
                                ;port pin before reading
        IN A, P1
        CPL A                   ;Convert data to high = true
        MOV R2, A
        MOV A, #02H
```

Figure A.3: Final Program for HPTEST-continued 2

```
        MOVD P6, A              ;Set NRFD and NDAC high
DAV:    MOVD A, P5
        ANL A, #08H
        JNT1 START              ;Reset if IFC low
        JZ DAV                  ;Loop if DAV still low
        MOV A, R2               ;Restore input to accumulator
        RET
;
;
;*************************************************************
; LISTEN reads data bytes and stores them in local RAM (up to
; 32 bytes).  Reading is terminated when a linefeed (LF) is
; encountered, at which time, the program returns to S2.  The
; terminating linefeed is saved as part of the message.
; Register R0 is used as the memory pointer, and R4 is the
; count of bytes received.  Only R0, R4 and the accumulator
; are affected.
;*************************************************************
;
LISTEN: JNT0 LISTEN             ;Wait for ATN to go high
        MOV R0, #018H           ;Initialize pointer
        MOV R4, #00H            ;  and counter
L2:     CALL GETB               ;Read a data byte
        MOV @R0, A              ;Store in memory
        INC R0                  ;Increment pointer
        INC R4                  ;  and counter
        ANL A,#07FH             ;Test lower 7 bits only
        XRL A, #LF              ;Compare with linefeed
        JNZ L2                  ;If no match, continue
        JMP S2                  ;If match, return to S2
;
;
```

Figure A.3: Final Program for HPTEST-continued 3

```
;************************************************************
; TALK transfers a block of bytes to HP-IB.  <R4> bytes,
; starting at the base address 018H (24D), are sent to the
; HP-IB bus.  At conclusion, R4 is zero.  R0, R4, and the
; accumulator are affected.
;************************************************************
;
TALK:   JNT0 TALK               ;Wait for ATN to go high
        MOV A, #03H
        MOVD P4, A              ;Set SRQ and talk enable high
        MOVD A, P6              ;Read NRFD and NDAC (throwaway)
        MOV R0, #018H           ;Initialize data pointer
T2:     CALL OUTB               ;Send data byte
        INC R0                  ;Increment pointer
        DJNZ R4, T2             ;Decrement counter
                                ; if not zero, send again
        MOV A, #01H             ;All bytes sent:
        MOVD P4, A              ;Set SRQ high, talk enable low
        IN A, P1                ;Read data lines (throwaway)
        MOVD A, P5              ;Read NRFD and NDAC (throwaway)
        JMP S2
;
;
;************************************************************
; OUTB transfers a single byte to HP-IB.  The byte to be
; output must reside at the location pointed to by R0.  Only
; the accumulator is affected.
;************************************************************
;
OUTB:   MOV A, #0CH
        MOVD P5, A              ;Set EOI and DAV high
        MOVD A, P6              ;Read NRFD and NDAC
```

Figure A.3: Final Program for HPTEST-continued 4

```
        ANL A, #03H             ;Test only NRFD and NDAC
        XRL A, #03H             ;If both high, no listeners:
        JZ OUTEX                ;  go to exit
        MOV A, @R0              ;Fetch byte for output
        CPL A                   ;Convert to high = true
        JNT1 START              ;Reset if IFC low
        OUTL P1, A              ;Send data byte (latched)
OUT2:   MOVD A, P6              ;Read NRFD
        ANL A, #01H             ;Test NRFD only
        JNT1 START              ;Reset if IFC low
        JZ OUT2                 ;Loop until NRFD low
        MOV A, #04H
        MOVD P5, A              ;Set EOI high and DAV low
OUT3:   MOVD A, P6              ;Read NRFD and NDAC
        ANL A, #02H             ;Test NDAC only
        JNT1 START              ;Reset if IFC low
        JZ OUT3                 ;Wait until NDAC high
        MOV A, #0CH
        MOVD P5, A              ;Set EOI and DAV high
OUTEX:  RET
;
;
;
END
```

Appendix B. Implementation of the Network Bus

The hardware description of Section III omitted any details of the network medium and the transceivers used to implement the bus. This appendix contains a brief description of three different network implementations and the advantages and disadvantages of each.

The first section of all of the transceivers is a Manchester encoder/decoder (Harris HD-6049) [20]. The encoder takes clock and data and produces a single stream of data from which both clock and data may be recovered, which function is performed by the decoder. The Harris integrated circuit performs both functions, requiring only an 11 bit delay for synchronization. The delay is accomplished by use of modem control lines RTS (Ready to Send) and CTS (Clear to Send). The BIU/CCU sets RTS to indicate it wishes to send data. The Manchester encoder responds by performing synchronization and then sets CTS to indicate that it is ready for actual data. Synchronization is only required once for each continuous message.

As originally designed, the network bus was implemented solely in fiber optics, capitalizing on the superior electromagnetic noise immunity of optics. However, the cost of such a network was much higher than a traditional wire network due to some necessary specialized technology. As a cost reduction measure, a second prototype network was designed with the same structure, only using wire pair as the transmission medium. While this design had a substantial price improvement, the potential harshness of a factory environment continued to cause concern. The final step, therefore was to design the network with a combination of both, offering wire pair economy in the majority of the network, but fiber optic reliability in the limited areas where electromagnetic noise was sufficiently

threatening. Each design is described below.

1. Fiber Optics Implementation

While several existing networks have employed fiber optics as the transmission medium (for example, NEC's N6770 Datalink [21]), very few of these are bus type networks. Most are ring type networks, allowing the optics to be used in a unidirectional, point-to-point configuration. Bidirectionality can only be achieved by some sort of tap or "T" connector; unfortunately, such devices for fiber optics tend to be rather expensive. More importantly, because optical signals are power as opposed to voltage or current, the amount of signal tapped off at a given node must be lost to downline receivers, constituting a sizeable attenuation problem in a multi-node network. Of course, repeaters may be employed to maintain more manageable signal levels, but in that case, the inherent reliability of the bus structure has been compromised. For the first design, it was determined to adopt a strictly passive optical bus structure, accepting the imposed limitations.

The limitations thus imposed acted to restrict the number of possible nodes, and to a lesser extent, the distance between nodes. The optical transceivers selected for the network, developed by Mr. Robert Harris of GE, had typical sensitivity of -50 dBm with a dynamic range of about 25 dB. Using an LED source at 820 nm, a typical value for the optical power coupled into the fiber (Siecor 133, 50 micron core) was -13 dBm (50 microwatts). The fiber had a typical attenuation of 10 dB per kilometer; an interstation distance of 100 meters was chosen for the prototype. Passive optical couplers were chosen which had a 1:1 tap ratio with a 1 dB insertion loss (insertion loss is the input signal less the sum of the output signals). Because the optical receiver and transmitter are separate units, a

coupler was required at each station to combine transmitted and received data to a single fiber. No additional couplers were required for the stations at either end of the bus, where communication is unidirectional, but an additional coupler was required at each mid-bus location, where signals are being received and sent in both directions. Finally, modular connection to equipment required a combination of fusion splices, which exhibited a worst case loss of 0.2 dB, and Amphenol connectors, which had a worst case loss of 2 dB. These factors combined to form the flux budget for the optical bus. Figure B.1 shows the configuration of one end station and one mid-bus station.

For the flux budget, there are two conditions which must be satisfied: first, the signal strength at any receiver must not be less than the minimum sensitivity, and second, the difference between the strongest possible signal and the weakest possible signal at any receiver must not exceed the dynamic range of the receiver. From the figure, it can be seen that any signal will encounter two connectors, but a varying number of couplers, fiber links, and splices. The lowest attenuation will occur when a signal is sent from an end station to its nearest neighbor; the highest will occur (for greater than three stations) when a signal is sent from one end of the bus to another. If there are N stations, those attenuations are given by:

$$2(2dB) + 3(4dB) + 3(.2dB) + 1dB = 17.6dB \quad \text{lowest}$$
connector coupler splice link

$$2(2dB) + N(4dB) + 2(N-1)(.2dB) + (N-1)(1dB) = 5.4N+2.6 \quad \text{highest}$$

The two conditions reduce to:

$$-13dBm - (5.4N + 2.6) > -50dBm$$
$$5.4N + 2.6 - 17.6 < 25dB$$

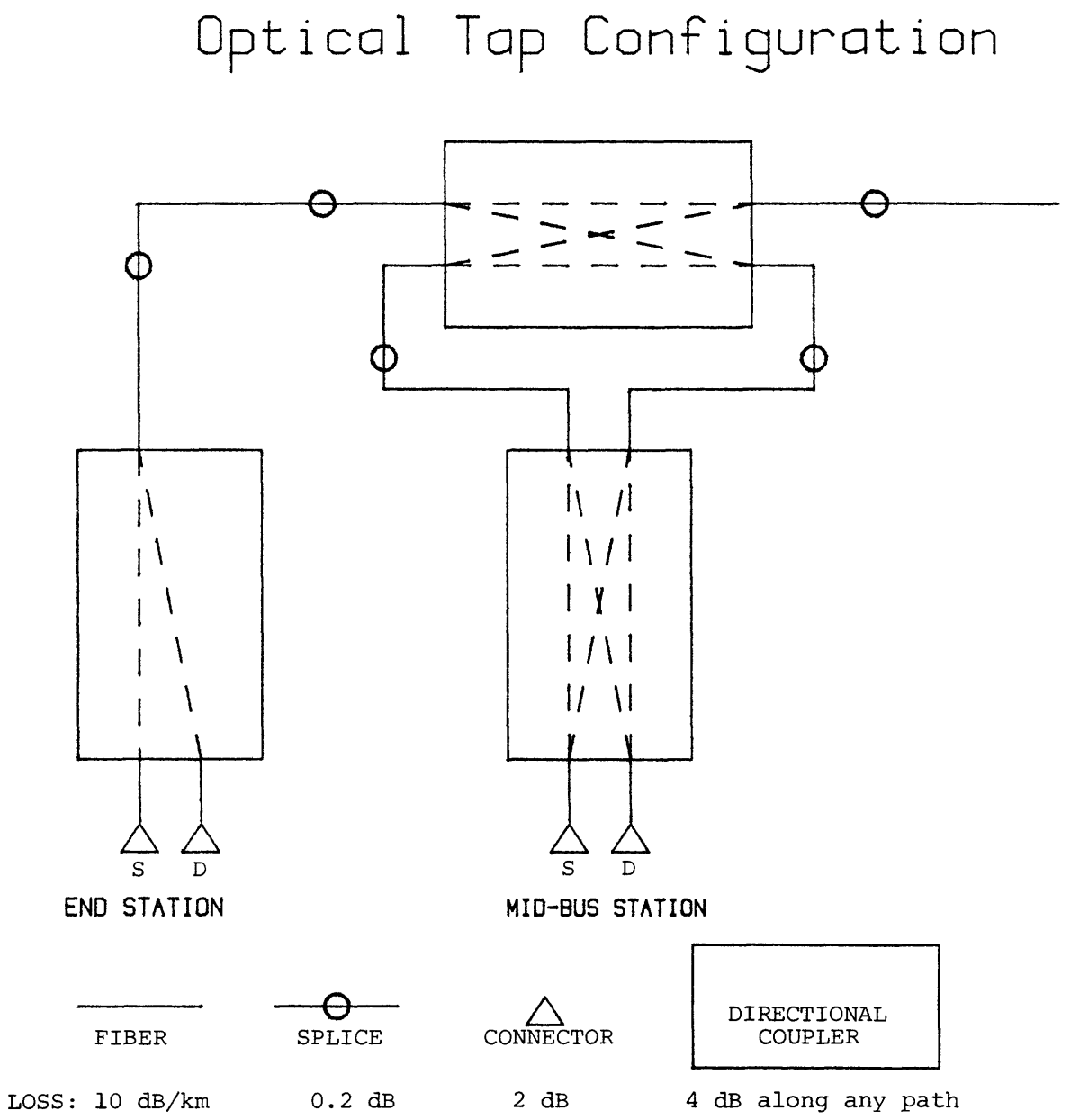


Figure B.1: Optical Tap Configuration

The first condition is satisfied for N less than 6.4, while the second is satisfied for N less than 7.4. N must be an integer, and the more stringent condition observed, thus the passive optic bus will support a 6 station configuration. With six stations, the margin for error is only 2 dB, and reasonable practice would suggest a maximum of five stations, giving a margin of 7.4 dB. Alternatively, the distance between stations could be shortened or lengthened, with a difference of 1 dB per hundred meters. Until cable lengths exceed an average of 400m, however, the coupler loss will be the more significant restriction on the number of stations.

Even accepting the restricted number of stations, however, the cost of the purely fiber optics network is prohibitive for a "low" performance system. The optical couplers mentioned above cost approximately $300 and the fiber optic cable on the order of $2 per meter. Adding the cost of connectors, the price per station approaches $750 for the transceiver alone (almost five times the price of the other hardware). Thus the cost needs to decrease dramatically in order to compete in the projected market.

2. Wire Pair Implementation

An economical alternative is twisted wire pair, using differential line drivers as the transceivers. Specifically, the transceivers chosen were for EIA standard RS-422. The cost of a transceiver (Texas Instruments SN75178) and supporting hardware is about $25, with copper wire pair less than $.15 per meter. Because of loading and transmission line effects, there are limitations to data rate, total network line length, and number of stations, but these limitations fall well within the projected network specifications. The maximum number of stations for RS-422 is 32. The data rate and network

line length are jointly constrained. The data rate may be as high as 10 Megabits per second if the line length is limited to no greater than 40 feet. The line length may be increased by a proportional decrease in the maximum data rate, up to a maximum length of 4000 feet at a data rate of 100 Kilobits per second. At the projected network data rate of 1 Megabit per second, the maximum network line length would be 400 feet, or about 120 meters.

Clearly, the network line length restrictions are the main disadvantage of the wire pair network. Furthermore, because the twisted pair is not shielded, there may be significant electromagnetic noise problems. While the line length restrictions could be overcome by forming several independent branches in the network, the interference problem would remain, and thus the wire pair network alone is not the final solution.

3. Multi-media Implementation

Many of the projected applications of the network involve a network configuration of several local clusters and one or more long interconnecting length. Clearly, the wire pair implementation is ideally suited to the clusters of such a network, because of the short distances involved, and the fiber optic implementation is more suited to the long lengths, which also are more likely to pass through areas of electromagnetic interference. This common topology suggests the use of a hybrid implementation, using both fiber optics and wire pair.

This multi-media implementation is accomplished by using the branch structure previously discussed. The network remains a single network, that is, there is still only one CCU and each BIU may send and receive data directly from all other BIU's, but the links interconnecting the network stations may be either fiber

optics or wire pair. The advantage is clear, combining the economy and larger number of stations of the wire pair implementation and the longer length and noise immunity of fiber optics. The cost increase over the wire pair implementation may be small, since many applications will require only the minimum of two optical stations.

In this implementation, there will be three types of tranceivers: fiber optics transceivers, wire pair transceivers, and hybrid transceivers which link a wire pair branch to a fiber optics branch. The wire pair transceivers are as described above for the strictly wire pair implementation and the fiber optics transceivers are the same in general structure as described earlier for the purely fiber optics implementation, except for some hardware improvements. To decrease transceiver size, an integrated fiber optic receiver (Burr- Brown FOR110KG) and an infrared LED (Motorola MFOE102F) replace the custom designed fiber optic transceiver. The Burr-Brown receiver is a zero-crossing receiver, requiring only TTL data and clock and power supply. The LED is capable of radiating 700 microwatts at a wavelength of 9000 Angstroms. The flux budget for this new transceiver is approximately the same as for the custom designed units; moreover, the entire transceiver fits in a 2" by 2" circuit board.

The hybrid transceiver must perform two functions: data must be be both exchanged with the BIU and forwarded to the remainder of the bus. For example, consider a hybrid station which receives data from its wire pair transceiver. This received data must be forwarded to the BIU and, in order to preserve the bus structure, the data must be transmitted to the fiber optic link. Thus, for received data, the hybrid station is a directional repeater. When the BIU transmits, both the fiber optic and the wire pair transceiver must transmit the data. Thus, for transmitting,

the hybrid transceiver is a bi-directional transmitter. This function requires additional logic, largely consisting of ORing the two receiver outputs to the BIU and allowing each transmitter enable to be activated by the BIU or the other receiver.

The main disadvantage of this structure is the repeater nature of the hybrid transceiver. In a sense, the advantages of the bus structure are compromised; if a hybrid station fails, all "downstream" BIU's will lose data. Furthermore, additional delays are added and the concept of simultaneous reception becomes unrealistic. Thus, the BIU's and the CCU must be programmed with short delays to allow all stations to have received all current messages. Fortunately, these delays are small (generally less than one bit period), but even small delays, as seen in Section V, will reduce performance.

As a final note, the multi-media configuration can be exploited to bring a further reduction in cost. If the fiber optic links of the network only connect two stations, then both transceivers are like the transceivers at the end of the network shown in Figure B.1. Each of these stations require only one of the expensive optical couplers, thus decreasing the cost per transceiver to about $450.

References

1. Mokhoff, N. "Local Data Nets: Untying the Office Knot." IEEE Spectrum, volume 18, number 4 (April 1981), p. 57.

2. Dovens, J. "Market Research Survey on Local Area Networks." Rockville, Maryland: General Electric Information Services Co., 1981, pp. 4,5.

3. Thurber, K.J. and Freeman, H.A. "Architecture Considerations for Local Computer Networks." Proceedings, First International Conference of Distributed Computing Systems, October 1979. IEEE, Inc., pp. 131-133.

   Clark, D.D., Pogran, K.T. and Reed, D.P. "An Introduction to Local Area Networks." IEEE Proceedings, November 1978, pp. 1497-1499.

4. Thurber, K.J. and Freeman, H.A. eds. "Local Computer Networks." New York: IEEE, Inc., 1981, section 6.

5. Thurber, K.J. and Freeman, H.A. eds. "Local Computer Networks." New York: IEEE, Inc., 1981, section 4.

6. Kuo, F.F., ed. "Protocols and Techniques for Data Communication Networks." Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1981, pp. 29,30.

7. Clark, D.D., Pogran, K.T. and Reed, D.P. "An Introduction to Local Area Networks." IEEE Proceedings, November 1978, pp. 1502,1503.

8. Clark, D.D., Pogran, K.T. and Reed, D.P. "An Introduction to Local Area Networks." IEEE Proceedings, November 1978, p. 1502.

9. Ketelhunt, W.J. "Communications System for Series 6 Process Control: Strategic Analysis." General Electric Industrial Control Division, Charlottesville, Virginia, July 1982 (internal document), p. 11.

10. Butler, H. "Local Area Network Technology Primer." GE Aerospace Electronics Systems, Utica, New York, January 1981, p. 4.

11. "IBM Synchronous Data Link Control General Information." IBM, publication number GA27-3093-1.

12. "MCS-48 Family Microprocessors User's Guide." Intel Corp, Santa Clara, California, 1980, p. 4.

13. Sources for remaining reference to Intel products:
Intel Component Data Catalog
Peripheral Designer's Guide
MCS-85 Family Microprocessors User's Manual
UPI-41A (8741A) User's Manual
all Intel, Santa Clara, California.

14. "MCS-85 Family Microprocessors User's Guide." Intel Corp., Santa Clara, California, 1981, pp. 3-1,3-2.

15. Davies, D.W., Barber, D.L., Price, W.L. and Solomonides, C.M. "Computer Networks and Their Protocols." New York: John Wiley and Sons, 1979, p. 50,51.

16. IEEE Standard-488 1978. IEEE, Inc. For a more detailed and readable description, see "Condensed Description of the Hewlett-Packard Interface Bus." Hewlett-Packard Co.,

Loveland, Colorado. Hewlett-Packard Publication # 59401-90030.

17. Data available from manufacturers. Of particular note, "uPD7201 Technical Manual." Nippon Electronics Corp., Natick, Massachusetts, 1981.

18. Vo-Dai, T. "Throughput-Delay Analysis of the Non-Slotted and Non-Persistent CSMA/CD Protocol." Local Computer Networks, P.G. Ravasio, ed. North Holland Publishing Co, 1982, p. 473.

Shachem, N and Hunt, V.B. "Performance Evaluation of the CSMA/CD Channel Access Protocol in Common Channel Networks." pp. 409-413. Local Computer Networks, P.G. Ravasio, ed. North Holland Publishing Co, 1982.

19. "8080/85 Assembly Language Programming Manual". Intel Corp., Santa Clara, California, p. A1-A4.

20. Harris Semiconductor Co. Data, 1980. See also Sanders, L. "Manchester II Transfers Data with Integrity, Speed." Electronic Design, March 19, 1981, p. 233-238.

21. "The NEC 6770 Datalink". Wellesley, Massachusetts: NEC Electronics, Inc., 1981.