SCI-net: A Very Low Cost CSMA/CD Network for Linking Small Microcomputer Systems

Rene Stanley Hollan

A Thesis

in

The Faculty

of

Engineering

and

Computer Science

Presented in Partial Fulfillment of the Requirements
for the degree of Master of Computer Science at
Concordia University
Montreal, Quebec, Canada

March 1984

ABSTRACT

SCI-net: A Very Low Cost CSMA/CD Network
for Linking Small Microcomputer Systems

Rene Stanley Hollan

This is a description of the design and implementation of a carrier sense, multiple access network with collision detection. Included are the basic design philosophy, operating restrictions and parameters, complete hardware and software descriptions, as well as data collected from actual operating tests and computer simulations. The network is unique in that all protocol support functions, mapped onto the first four layers of the International Standards Organization Open Systems Interconnect model are implemented in software, including collision detection.

ACKNOWLEDGEMENTS

The author wishes to acknowledge the following people for their significant contributions to the development of SCI-net: Dr. Terrill Fancott, for the concept and system specifications of SCI-net; Amitava Sen, for the development of an early prototype version of the hardware interface; Mark Weiser, for assistance with software and hardware development; Jacques Blaison, for assisting in the procurement of components and prototyping boards; and Frank Maselli, for his assistance during the early stages of the implementation of the network design. In addition, the critical comments and criticisms of Eamon Egan and Dr. Terrill Fancott were greatly appreciated. The author also wishes to acknowledge the financial support provided by the Natural Sciences and Engineering Research Council.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.0 INTRODUCTION

Local network architectures using CSMA/CD link level protocols present the attractive property of fully distributed control. The success of the Experimental Ethernet and it's subsequent commercial implementations has stimulated the development of many CSMA/CD systems with similar characteristics, commonly referred to as 'Ethernet-class' systems [SHOC82]. High performance systems, using coaxial cable as a transmission medium, offer raw transmission speeds of up to 50 megabits per second [FRAN82], while the Ethernet specification provides a 10 megabit rate for 500 metre segments [THUR82]. This performance is, however, bought for a price. Contemporary Ethernet per-station costs, including interface and tap, vary from $1500 to $3000. While this is not expensive for networks of high performance computers, it is not reasonable for linking personal computers whose total

cost is in the same order of magnitude.

The need for a low cost networking technology has resulted in the development of a class of Ethernet-like systems based on standard LSI communications devices, and using low cost media such as twisted pair or twinax [THUR82]. Some of these systems, such as the Corvus Omninet or the Cromemco C-net have achieved raw bit rates between 250 kilobits and one megabit, while holding interface prices to about $1000. Similarly configured research systems based on the Motorola Advanced Data Link Controller (M6854) report speeds of 150 [SMIT82] and 500 [HUTC82] kilobits for relatively low component cost.

This thesis presents the design and implementation of a very low cost CSMA/CD system. At a raw bit rate of 125 kilobits per second, the system has sufficient performance for a medium sized office or laboratory environment, in particular when the usage statistics on the Experimental Ethernet reported by Shoch and Hupp [SHOC80] are considered. Although the theoretical channel utilization for their system is 97%, they reported an average usage of 0.60% to 0.84% over a 24 hour day for 120 users on a 2.94 megabit per second line. These figures suggest that for a large class of applications a significant cost-performance tradeoff is

justified. In our design we push this tradeoff to the limit, moving as many functions into software as is feasible without sacrificing the performance of the individual processors of the network. The result is an interface design only marginally more complex than a standard RS-232C interface, using less than $15 in components, but fully implementing a CSMA/CD protocol.

Since the primary goal is the development of a low cost system, standard off the shelf LSI components are used in a hardware design that requires only seven integrated devices. The system developed is unique in that collision detection, traditionally performed by hardware, is fully implemented with software. Although this results in delays in detecting message collisions and, as shall be illustrated later, reduces the effective bit rate of the channel when collision detection is in effect, the design presented reduces the overhead incurred to an acceptable level.

Simulations are performed comparing software implementation of key functions such as collision detection and carrier sensing with hardware implementations of the same. The latter are simulated by scaling down the operating speed of Ethernet to that of the software system while retaining relative timing relationships within the protocol.

Although the system is almost entirely software based, the primary limitation upon raw bit rate is determined by the choice of hardware components used. However, faster hardware would not make a significant difference in the maximum atainable raw bit rate unless processor speed were increased accordingly. At present only a two-fold increase increase in processor speed can be obtained.

CHAPTER 2

A BACKGROUND DISCUSSION OF NETWORKS

2.0 INTRODUCTION

A computer network is a means for interconnecting two or more computing devices to allow the transfer of information between them. The desire to interconnect several different computer systems via a single network has given rise to some trends and standards governing networks.

This chapter describes some of these standards and examines some common networks in existance today.

2.1 DEFINITIONS

In order to be able to make useful comparisons between different networks, a fundamental understanding of the terms used to describe and define these networks is essential. Thus, terms such as protocol, raw bit

rate, line utilization, protocol overhead, and topology are explained in detail before proceeding any further.

2.1.1 PROTOCOL

A protocol serves to define the means by which a device communicates with other devices on a network. A protocol is usually expressed in terms of its syntax, semantics, and timing.

A protocol's syntax refers to the physical nature of the data sent along a network. At various levels of abstraction it defines the type of electrical signals used, or the ordering of various fields of information in packets or frames, these being the smallest unit of data sent from one station to another. Such information often includes, but is not limited to, source and destination addresses, the type of the packet, the nature of the data being sent, and the length of the packet.

The semantics of a protocol define how the information passed between devices, or stations, is to be interpreted. In contrast to the syntax, which specifies the manner in which data is formatted, the semantics describe how the various pieces of

information are to be used in supporting the protocol. For example, the type field of a packet may determine whether the data is to be interpreted as protocol management information, or data that is actually being passed between stations via the particular protocol in use. A 'source address' field in one protocol may have an entirely different meaning in another protocol. Furthermore, certain signalling functions such as checksum errors, and packet acknowledgements, can have vastly differing implications in different protocols.

In short, the syntax describes the way in which information is ordered and formatted, while the semantics describe the way in which this information is to be interpreted and used.

A necessary requirement of any protocol is the timing of the protocol. Besides having to specify the timing relationships between successive bits of information, the protocol may place restictions on the timing relationships between successive packets, the maximum allowable time between transmitting a packet and being able to receive the next one, or the minimum allowable time between successive packet transmissions.

For example, the Ethernet protocol specification indicates a serial transmission rate of 10 megabits per

second with no less than 9.6 microseconds between successive packets. [SHOC81]

2.1.2 RAW BIT RATE

The maximum rate at which a network can support the transmission and reception of data is referred to as the raw bit rate. It is the theoretical limit to the speed at which information can pass from one station to another.

Consider Ethernet, with a raw bit rate specification of 10 megabits per second. The actual rate of transmission of information is, in fact, significantly less. Inter-packet spacing, protocol-related information within packets, and various other delays reduce the actual speed at which data can be sent. The same is true of any other protocol since there is always some overhead in arbitration between stations competing for access to the network medium.

A special case of a network protocol is a bus protocol, that is, the method used to transfer data along a parallel bus as opposed to a single serial line. In this case, the maximum bit rate specification

becomes a misnomer since data is, in fact, being transferred several bits at a time. It is more correct to refer to the parameter as bus bandwidth. This is defined as the capacity of the bus to transfer information. It is still expressed as a quotient of information transfer over time, but the speed requirements of any single bus line are reduced by a factor proportional to the number of data transferring lines in the bus.

2.1.3 LINE UTILIZATION

The line utilization of a particular network is a percentage of the maximum bit rate. It indicates the network's capacity to transfer information, both station-to-station data as well as protocol management information. Factors that contribute to this percentage are turnaround time between sending a message and being able to send the next one, network access arbitration, message length, and the number of stations on the network.

Since the line utilization is dependent on a large number of factors, it is usually expressed under some fixed conditions, such as 'at saturation'. This indicates a measure of the line utilization when all

stations wish to use the network. Typically, the line utilization is an asymptotic function of the number of stations transmitting.

When the line utilization is measured with only one transmitter and one receiver, it indicates the extent to which network arbitration affects performance. The greater the effect, the lower the number of stations that can be adequately supported, all other factors notwithstanding.

2.1.4 PROTOCOL OVERHEAD

The protocol overhead is a simple measure of what percentage of data being transmitted in a packet is for the management of the network protocol. Since this factor directly affects the effective transmission rate from one station to another, it is important to keep it as low as possible. In most protocols, the protocol support information in a packet is always of a fixed length while the protocol may support varying length data fields. Clearly, the greater the packet length, the lower the protocol overhead.

In addition, longer packets imply less arbitration time and may improve line utilization as well. This

apparent improvement does not, however, come without a price. When packets are too long, the average time required to access the network increases and the law of diminishing returns takes effect.

Selecting an appropriate packet length is thus an important factor in the fine tuning of a network.

2.1.5 NETWORK TOPOLOGIES

The topology of a network indicates the manner in which stations are connected to the network. Ring, star, and other topologies are all common. A ring topology requires that all stations be connected in a ring, that is a closed loop including all stations. Star networks employ a central network controller or dispatcher that all stations communicate with as an intermediary when sending information to other stations. In addition, the use of repeaters or routing stations affects the manner in which stations can communicate and thus is descriptive of one aspect of the network topology.

The topology of a network places certain restrictions upon the types of protocols that are supported. A network with several message routing

stations, for example, might require routing information to be present in packets that circulate around the network. The possibility that some packets might circulate forever must be contended with by the protocol, as well as other topology dependent aspects.

2.2 ISO LAYER MODEL

The International Standards Organization, in an effort to produce an abstract model of network systems introduced the now familiar protocol layer model. The concept of layers of a protocol should follow naturally from the preceding discussion: Protocol layers are analogous to the various levels of abstraction one faces when studying networks. Although designed to describe communications in the long-haul environment, the Open Systems Interconnection Model is also applicable to local area networks.

2.2.1 ISO MODEL OVERVIEW

The ISO Open Systems Interconnection Model [ISO] (henceforth referred to as 'the ISO model') considers levels of abstraction within network systems as layers. Each layer considers a particular aspect of information

transfer from the lowest physical layer to the applications layer.

Relating all networks to a common model facilitates the comparison of such systems and indicates at what level the end user must deal with the system: either direct hardware interfacing, application programs (such as a distributed database query system), or some degree of abstraction between these two extremes.

The ISO model is composed of seven distinct protocol layers. These layers have been designated 1) the physical layer, 2) the link layer, 3) the network layer, 4) the transport layer, 5) the session layer, 6) the presentation layer, and finally, 7) the application layer. The advantages of this modular, layered, system, are clear:

1) Facilities necessary for the proper functioning of one layer can be made invisible to other layers. In particular, a given layer N need only interface to layers N-1 and N+1 through a set of standard system services.

2) Complex systems can be broken up into manageable

sub-systems.

3) Changes to a layer are, in most cases, transparent to other layers. Thus, maintenance of the system is simplified.

4) Services provided by a particular layer N can make use of all functions offered by lower layers. A strict ordering of layers need not be adhered to, although this offers some benefits.

5) Several implementations of a given layer N can co-exist within the system. In particular, there may be several application level interfaces that provide different functions to the user.

6) Layers can be modified, or deleted altogther (the model permits layers to be empty) as required throughout the evolution of the system as required. Fine tuning of a system and even major protocol changes can thus be accomplished with a minimum of trouble.

7) Each layer may be tested and verified independently of other layers. As stated earlier this increases confidence in the reliability of the overall system. Program verification is an

ongoing area of research and proving correctness of network systems is not trivial.

Although the layer model applies a rigid, modular, structure to network systems, the interpretation of just what services are to be provided by a particular layer are dictated primarily by the subjective opinions of the network designer. Few standards exist, and while the ISO Model provides a convenient means to describe network facilities, it remains to be seen whether standard model layer functions and interfaces will be formed.

2.2.2 THE PHYSICAL LAYER

The physical protocol layer defines the electrical and mechanical specifications of the network. The physical layer performs the transmission and reception of unstructured streams of data bits over the network communications medium.

Common physical layer protocols include the standard RS-232C, RS-442, CCITT X.21 protocols, as well as the myriad of proprietary protocols designed to interconnect various manufacturers' computing

equipment.

2.2.3 THE LINK LAYER

The link protocol layer (also known as the data link layer) performs the transmission and reception of structured streams of bits over the network media. The raw bit stream handled by the physical layer is sub-divided by the link layer into structured frames or packets. Reserved bit patterns, reserved character sequences, or explicit bit counts serve to delimit packets from one another.

Some means of error detection, such as a packet checksum, cyclic redundancy check, or parity bit schemes, are usually implemented in the link layer.

Some standard link layer protocols currently in use include HDLC, SDLC, ADDCP, BiSync, and Ethernet's CSMA/CD sceme.

2.2.4 THE NETWORK LAYER

The network layer handles the transmission of a packet from a source node to a destination node. Since

this may involve intermediate routing stations and alternate and/or preferred transmission paths some strategy is required to properly send a packet from source to destination. The network layer provides the mechanism to implement this strategy.

Note that local area networks do not usually have routing or store-and-forward stations. The network layer can thus be very small or, as is often the case, totally empty.

2.2.5 THE TRANSPORT LAYER

The transport layer provides reliable end-to-end or host-to-host communication over the network. The various schemes used to accomplish this can be divided into two categories: virtual circuits and datagrams. Virtual circuit systems set up logical or physical connections between network stations to allow apparently closed communications links to be set up. Datagrams include as part of their packets routing information which indicates the source and destination of the packet as well as any routing information. Note that datagrams can be used even if no routing is performed (and hence no routing information need be provided).

Virtual circuit protocols have the distinct disadvantage of imposing an overhead in the protocol required to set up the virtual circuits before communication between stations can take place. Despite this disadvantage long-haul systems traditionally employ virtual circuit protocols. This is due to the fact that they are usually easier to implement. [THUR82]

Although more difficult to implement on long-haul, store-and-forward networks than virtual circuit protocols, datagram schemes are easier to implement on local area networks. Thus, one should expect them to be the dominant type of transport layer protocol in future LAN implementations.

2.2.6 THE SESSION LAYER

The session layer manages end-to-end communications between processes running on network stations. Typically it involves the translation of logical process or port names (character strings) into logical and physical unit numbers and network addressing information. The opening and closing of network transfer links and generation of datagrams as well as managing virtual circuits is all performed by

this ISO Model layer. Of course, the actual functions performed by this layer depend on services provided by the transport and lower layers, as well as local operating system standards and interfacing conventions.

2.2.7 THE PRESENTATION LAYER

The presentation layer is responsible for the translation or transformation of data to be sent to the session layer. For example, a text compression scheme may be employed to reduce the volume of data sent along the network. The presentation layer would then be responsible for the compression and expansion of information sent to and received from the network.

2.2.8 THE APPLICATION LAYER

The application layer provides a variety of application-specific protocols to application programs running on the various stations scattered throughout the network. Such applications include, but are not limited to, electronic funds transfer, mail, distributed database systems, remote job entry, and registration and reservation systems.

As of this writing there do not exist any standard application layer protocols and, due to the diversity

of potential network applications, it is unlikely that there ever will be.

2.3 SOME LOCAL AREA NETWORKS

To familiarize the reader with the types of networks currently available several will be examined. Due to its success Ethernet deserves close examination.

Developed by Xerox, Ethernet is a local area network employing a coaxial cable with 50 ohm characteristic impedance as the network medium. Data is transmitted along this medium at a rate of ten megabits per second in a Manchester encoded format. This results in a 50% duty cycle and insures a transition in the middle of every bit cell. [SHOC82] The first half of the bit cell contains the complement of the bit value and the second half contains the true value of the bit.

Packets are divided into the following fields: a 64 bit preamble consisting of alternating one and zero bits with the exception that it ends with two successive one bits. Following the preamble is a 48 bit destination address (the least significant bit is transmitted first, by convention), a 48 bit source

address, a 16 bit type field, from 46 to 1500 data bytes (each byte is eight bits wide), and finally, a 32 bit cyclic redundancy check.

The destination address specifies the intended recipients of the packet. If the first bit is zero then the field specifies a unique destination address. If it is one then the remainder of the field specifies a logical group of destination addresses. The special case of all ones indicates that the packet is broadcast to all stations on the network.

The type field specifies which higher level protocol is being used. The variable length data field contains the information that is actually being sent from one station to another. The specification of a minimum 46 byte data field ensures that valid packets will be distinguished from collision fragments.

The cyclic redundancy check (CRC) is computed over the destination address, source address, type field, and data field. This sequence of bits (the first bit of the destination address is taken as the most significant bit) is divided by a specific generating polynomial G(x) to yield a remainder R(x). The bit pattern corresponding to R(x) is transmitted as the CRC.

Ethernet specifies a minimum inter-packet spacing of 9.6 microseconds with a maximum end-to-end round-trip delay of 51.2 microseconds. This places a restriction on the length of cable that can be used and hence the maximum distance between any two stations on an Ethernet network. Furthermore, any packet shorter than the minimum valid packet length is discarded as a collision fragment.

The following control procedure defines how and when a station may transmit packets on the common cable. The purpose of this scheme is the fair resolution of contention for the network among transmitting stations. A station is in DEFER mode when there is a carrier present on the network or the minimum packet spacing time after a carrier has been lost has not yet elapsed. A station may enter TRANSMIT mode, and hence send data, if it is not in DEFER mode. It may continue to transmit until either the end of the packet has been reached or a collision is detected. If a collision is detected the station enters ABORT mode; transmission of the current packet terminates and a jam of four to six bytes of arbitrary data is sent to ensure that all other transmitting stations also detect the collision. After a station has detected a collision and enters ABORT mode, it waits a random retransmission time defined by the following Truncated

Binary Exponential Backoff algorithm. The delay before the n'th attempt is a uniformly distributed random number in the range [0, 2**n-1]. For attempts eleven through fifteen, this number is truncated to the range [0, 1023]. After sixteen transmission attempts a higher level protocol decision is required between either attempting further retransmission or abandoning the effort. This algorithm will resolve fairly contention among up to 1024 stations.

Although Ethernet is in widespread use, per-station costs are typically $1500 to $3000. Since the cost of small microcomputers is of the same order of magnitude, this may be considered excessive in many potential applications, especially when small numbers of stations are to be networked and the 10 Mb/s bandwidth is not required.

There are a host of other networks besides Ethernet available. The Magna III Cluster, manufactured by A. B. Dick Company [THUR82], for example, is based on a loop topology. Up to 255 stations can be connected to a single loop with as much as 1500 feet between stations. Although this network does support such diverse applications as word processing and electronic mail, and interfaces with CP/M, only Magna III devices are supported. The data

rate is a modest 100,000 bits per second. Multiple loops can be interconnected by means of loop gateways.

An interesting concept for a network was developed by Antel Systems Corporation. [THUR82] The physical medium in this network is simply existing AC wiring. Thus, no new cabling need be run when a network such as this is installed. Despite the low data rate of 300 bits per second (modulating a high frequency carrier which is superimposed on the 60 Hz AC line), such a system would appear to be ideal for low volume traffic: up to 2000 stations can be supported over one hundred thousand square feet without the need for signal redistibution. However the cost of the central controller ($29,500) is completely unacceptable for the large majority of applications.

It is clear, then, that low data rate does not imply low cost. The esoteric nature of the network medium of some networks often dictates the use of expensive technology. Truly low cost network systems are few and far between. This is ironic since the majority of network applications such as small commercial and educational networks do not require the support of large numbers of stations, or great bandwidth, and hence such networks should be relatively inexpensive to implement.

An exception to the high cost network is C-NET, a CSMA/CD protocol network using standard twinax cable, offered by Cromemco Incorporated. Boasting an respectable raw bit rate of 880 kilobits per second, C-NET supports as many as 255 stations up to 2000 meters apart. C-NET will work in a hostile electrical environment, unlike many other networks. The cost per station is a surprising $500.

This thesis introduces SCI-net (Software Controller Interconnect), a very low cost CSMA/CD network with software implementation of all protocol control functions. As will be seen, to realize per-station interface costs of approximately $100 (estimated retail cost) requires minimizing hardware as much as possible, using conventional network media, and moving as many functions as possible into software. SCI-net is unique in that both carrier sensing and collision detection are performed entirely in software.

CHAPTER 3

SCI-NET: A SOFTWARE CONTROLLED CSMA/CD NETWORK

3.0 INTRODUCTION

One of the main objectives in the development of SCI-net was keeping per-station costs as low as possible. This was achieved via compromises in raw bit rate and network performance against hardware complexity. The general idea is to place as many network functions as possible in the network software thus simplifying the hardware considerably. SCI-net is perhaps an extreme example of the application of this philosophy: only seven standard integrated circuits are required to implement a complete SCI-net interface.

The use of a twisted-pair network medium to support a moderate data rate of 125 kilobits per second further reduces installation costs as expensive cable is not employed. An added advantage of using a low data rate is the relative insensitivity of the network

to signal propagation delays. At the present data rate signal degradation due to cable loss is the most significant factor governing the maximum distance that can exist between two stations on SCI-net.

3.1 FACTORS GOVERNING SCI-NET DESIGN

Since hardware complexity and cost were to be kept to a minimum, standard off the shelf LSI (large scale integration) circuits were employed in the circuitry as opposed to custom devices. While certainly contributing to low design and per-unit cost, this decision is what renders collision detection via software a non-trivial task. Since the devices employed were not specifically designed for local area network applications, making them suitable to such applications required strong software support.

Despite the reliance of SCI-net on the performance of key functions by software, it is nevertheless important not to compromise raw bit rate any more than necessary. As a result the initial release of software to support the SCI-net interface does not display a distinct separation of ISO open model protocol layer functions. Rather, machine code timing is the governing factor in dictating just how the various

software functions are implemented and interfaces.

Furthermore, the overhead of the software required to handle the transmission and reception of messages over the SCI-net twisted pair medium must be kept to a minimum. This requirement directly affected the decision to use a hardware interval timer to assist in determining when a particular station is to attend to the network. It is clear that messages not intended for a particular station should be recognized as such as soon as possible so that the remainder of such messages can be ignored. The main purpose of the interval timer is to determine, once software has stopped the monitoring of the network, when the network is to be monitored again. In heavy network traffic this significantly reduces the overhead on any particular local station as only those messages that are destined for that station need be received in full.

SCI-net has been shown to operate very efficiently in a single-process environment. Particular network management functions, for example the transmission of a packet, require the complete attention of the local processor. When a single process is running, clearly it is the one that requested the transmission and thus must wait on the request being completed before being able to continue. Since the process is blocked pending

the completion of an I/O request the complete use of the processor to service that request is not detrimental to the system. However, this would certainly not be the case when several processes would be running since a single SCI-net I/O request would block all processes. The present implementation of SCI-net software would therefore result in a performance reduction directly proportional to the number of messages being sent, and the overhead of link access under heavy traffic conditions.

Since SCI-net is designed as a very low cost local area network, this factor is not unreasonable. A low cost network would normally be used to link small, single user systems. Larger systems would probably justify the increased expense of a more sophisticated network.

Nevertheless, this thesis will show how efficient networks can be derived from the SCI-net concept to efficiently serve multiple process stations without compromising implementation cost.

3.2 ISO LAYER DESCRIPTION

While the SCI-net software does not clearly

reflect the different layers of the protocol used (in the interest of speed), SCI-net can certainly be described in terms of such layers. The architecture of SCI-net has been developed over the first four layers of the ISO reference model. The only exception to this is an empty network layer. However, since there are no intermediate nodes through which packets must pass, an empty network layer is to be expected. This is one of the characteristics of a shared medium system.

The following discussion, therefore, examines SCI-net as a particular realization of an ISO reference model of a network.

3.2.1 SCI-NET PHYSICAL LAYER

As previously stated, SCI-net employs a single twisted pair of conductors as the physical medium through which packets pass from one station to another. This medium accomodates the maximum 125 kilobits per second raw bit rate of the network and has been demonstrated to support over 1/2 mile between stations in similar applications. Testing has shown that although the communications interface drives the network asymetrically in the 1 and 0 states, no negative effects from reflections were observed.

In addition to the network driver proper, the SCI-net interface includes an asynchronous communication interface adapter to support serial transfer of information over the network as well as an interval timer. The interval timer provides timeouts for the various software functions implementing the SCI-net protocol. This relieves the burden of timing functions from software and contributes significantly to efficiency and simplicity of operation: software timing functions are 100% processor intensive, a situation that must be avoided.

The SCI-net hardware is more fully described in chapter four.

3.2.2 SCI-NET DATA LINK LAYER

The data link software controls the transmission and reception of packets to and from the network. Software is also responsible for the carrier sense and collision detection functions. Successful transmission and reception of packets is assured through the use of an acknowledge protocol.

Messages are received and held for further processing in a message buffer. As a result, the

current running process need not be synchronized to incoming messages and can consume them at any rate. If the message buffer is full when a new message is received the new message is discarded and no acknowledgement is sent. After a suitable timeout interval the message will be resent. Eventually the message buffer will be consumed, the new message will be transferred to it, and reception will be acknowledged. Currently, transmitting stations are not aware of the speed at which messages can be consumed by receivers. Were the receiver to indicate this to the transmitter, then needless transmission could be avoided and use of the line would improve. This is left for future enhancements and upgrades to SCI-net.

Figure 3-1 illustrates the message format.

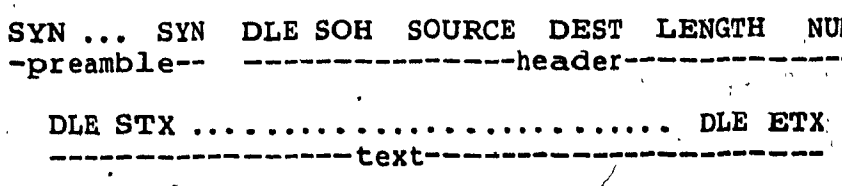


Figure 3-1

Currently, the data link software does not perform any error checking. This has been left for future implementation in the session layer. Such a situation should not surprise the reader, who will realize that such a deferment allows the use of several different

error detection methods without requiring modification of the basic SCI-net software. Since SCI-net is currently in an active research state, this is a logical choice to make.

The design of the message format is the key to complete implementation of the protocol in software. The preamble of SYN bytes insures that all receiving stations are fully synchronized, even in the presence of noise. Furthermore, they allow sufficient time for a station to respond to incoming messages.

A source address byte serves to identify the sender of the message. This byte supports collision detection. In the event that two stations commence transmission simultaneously the senders can detect the collision by comparing their source bytes with the bytes sent on the line. Due to the use of open collector network drivers it is possible that only a single station will detect a collision. This is inconsequential since the rest of the packet being transmitted will either have not been corrupted, if both stations transmit at exactly the same time, or will have been corrupted beyond the point at which it may be received in an incorrect state. In the latter case no receiving station will detect a valid header and so they will simply discard received bytes that

cause SCI-net interrupts. Although this occurrence is not totally avoided, the probability of it occurring is extremely small. The collision detection algorithm serves to reduce the number of timeout and resend attempts due to collisions. The network protocol would continue to function in the absence of such a collision detection algorithm although performance would be significantly compromised. Early slotted-ALOHA used such a scheme.

At this point it is appropriate to describe the various means used to ensure efficient performance. First, when a station has ascertained the network to be free (by monitoring it for a short time), it starts to transmit. However, this is not an indivisible operation in the global sense of the network. There exists a 'collision window' during which time another station may have already started to transmit. Since collision detection takes place only during the transmission of a packet header, this window must be smaller than the time required to transmit such a header. In fact, the smaller the collision window, the lower the probability of a collision occurring since a greater proportion of would-be collisions are avoided. The collision window is presently slightly longer than the time required to transmit a single byte at full speed. Although collisions will not be detected until

after the collision window has passed, stations testing the network to determine its status will find it busy one byte time after transmission begins and will have entered their receive service routine.

If two stations do start transmitting within the collision window then one of two things may happen: both detect the collision and back off, or one detects the collision and backs off.

In either case, if the transmissions were perfectly synchronized (an unlikely occurrence), all receivers will be pending on the completion of the packet. As soon as one of the two stations times out and retransmits an ill-formed packet will be received and discarded. Since this is an undesirable situation it is fortunate that the probability of its occurence is practically negligeable. Were the transmission not perfectly synchronized, as is the case with almost all collisions, all receivers would immediately discard the message fragments. If a single station did continue to transmit, its message would simply be discarded. While this is not an ideal situation, the overhead incurred is slight since a non-receipt of acknowledge timeout has been traded for a collision timeout.

The second factor that contributes to improved

network performance is the employment of a random collision timeout. Random timeouts are generally found to be better than constant ones. Consider the case when a fixed collision timeout is implemented: All stations whose transmissions collide with others will detect these collisions at the same time, time out for the same interval, and start retransmitting at the same time. As a result the consequence will be more collisions! Furthermore it is quite possible that such stations will continue to generate collisions ad infinitum thereby causing themselves to enter infinite loops and depriving other stations from transmitting in the newly created 'collision interval'. Random collision timeouts prevent this since such synchronization will be eliminated before it can start. Two synchronized collisions will, in fact, be quite rare. The reader should note that there is a very close relationship between the size of the collision window and the distribution of the random collision timeout intervals.

A third factor relating to network performance is the number of times in a packet that the occurence of collisions is tested. Although not significantly affecting data transmission rate in proper hardware implementations of collision detection, this has a dramatic effect on transmission rate when implemented

in software. When testing for collisions, and using common off-the-shelf serial communications components that nearly almost always employ double buffering, this double buffering must be effectively eliminated since it does no good to test the current byte against the last byte sent. A direct consequence is a pause in the transmission of data. Since the SCI-net protocol needs only to check one byte for possible collisions this effect is reduced as much as possible. Checking more bytes would not reduce the number of collisions, and would have a detrimental effect on network performance.

The receive protocol is designed to use as little of the processing time of the local processor as possible. In an ideal situation, the local processor should need to examine the line only when it is active and then only during the transmission of the header. This is accomplished by having a receive function design which incorporates several distinct modes of operation (Figure 3-2). In the line monitoring mode the ACIA interrupt is enabled, and the processor continues servicing the user's current process. The first byte in a packet to be received by the ACIA causes an interrupt, switching the processor to the header receive mode. This first byte is read and discarded since its purpose is to clear any pending status bits and received data in the ACIA registers.

From this point on the ACIA is synchronized to the incoming data stream. The header receive software monitors the line until it receives the unique sequence SYN DLE SOH. If this sequence is not properly detected, due to noise on the line, or, more likely, a collision, the packet header fragment is discarded and the processor returns to line monitoring mode, and the user task at hand. If however, this sequence is received, the processor proceeds to read the SOURCE, DEST, and LENGTH bytes. If the destination byte does not match the station then the message is ignored. The message may also be ignored if the station is not responding to the source of the message or the receive message buffer is full. In any of these cases the LENGTH byte is used to determine the length of time the station is to ignore the network. A timer is programmed to generate an interrupt after the specified time has elapsed, and interrupts from the ACIA are re-enabled. While the timer is counting down, control is, of course, returned to the user process, although the processor is not in line monitoring mode.

If the local station is capable of receiving the message, NUM is checked to see if it is a duplicated message. Consider the following case: Station A transmits a message to station B. Station B receives the message but the acknowledgement to station A is

lost. As a result station A retransmits the message. Station B detects that it is the same message by examining the NUM byte, and simply sends back an acknowledgement to replace the previous one. Notice that this requires a separate temporary NUM byte buffer for each station with which communication takes place. Currently SCI-net only supports one such buffer and so, a station may communicate with only one other station at any particular time. The preceding does not apply, of course, if reception of multiple messages can be tolerated, or can be handled at a higher protocol layer. SCI-net provides the facility for handling this situation, to some degree, in the data link layer.

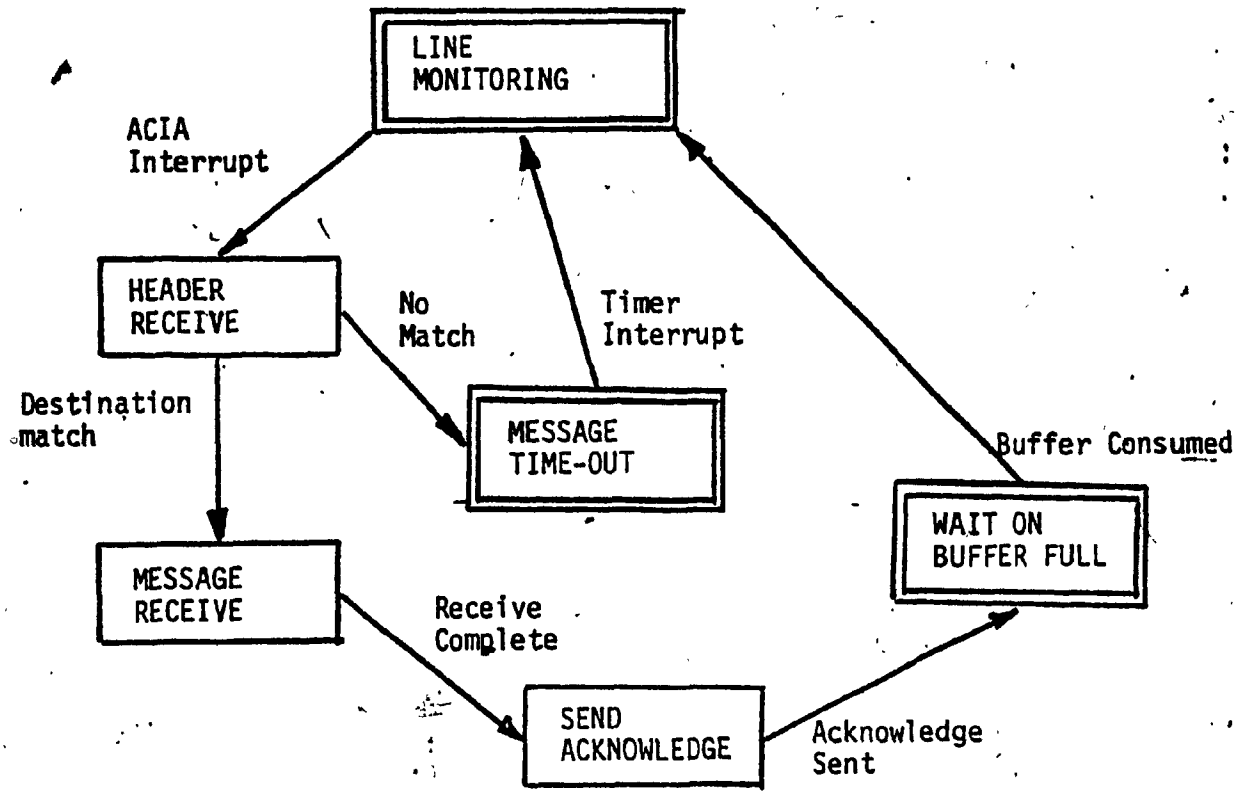


Figure 3-2

All successfully received messages are acknowledged by a special acknowledge message, illustrated in Figure 3-3. This is transmitted by the send function.

SYN ... SYN DLE SOH SOURCE DEST LENGTH NUM

DLE STX DLE ACK DLE ETX

Figure 3-3

The send protocol implements carrier sensing and collision detection. Once the send routine is entered, it will continue to attempt to send a message until it finally succeeds. In contrast to the receive function, this philosophy does not permit local processing while the send routine is waiting on a busy line. The increased complexity of including a 'read while send' function to accommodate this was not justified by the minimal increase in local processing it would allow, considering the fact that each station will normally send only a small proportion of the total messages on the system.

The carrier sense function is indicated by the ACIA read buffer full flag. To avoid any possible noise effects on the line, the receive buffer contents are first read and discarded to reset the flags. The buffer full flag is then inspected after a random delay. If the line is not busy the transmit routine proceeds to send. This initial random delay is to ensure that an equal chance is given to all stations to access the line. It is significant in the case of heavy traffic, where many stations may be waiting for

the end of a transmission. If the line is busy, it is actively monitored until it is free again. When it is free, the send routine transmits the header. When the SOURCE address is transmitted, it is read back from the receive buffer and compared with the station address. A collision will either corrupt the byte sent or result in it not being received at all. Thus, a hard coded timeout is in effect while waiting to read back the SOURCE byte just sent. If it is not the same as the station address, a collision has occurred, and the routine executes a random delay before retrying. In the case of no collision, the full message is sent, and the routine enters a wait for acknowledge timeout. If no acknowledge is received before the end of the timeout, the message is resent. The number of retries before signalling error to the local host is programmable.

This data link software is capable of sending and receiving continuous bytes at the full 125 kilobits per second speed. The system is fully asynchronous, in the sense that the individual stations need no external signal to synchronize themselves on either a bit, byte, or message level. Furthermore, if they enter the line monitoring mode at any point in a message transmission, they will synchronize themselves at the start of the following message. Incomplete message segments such as

collisions are discarded. Since it is not possible to send the message header without first establishing a quiescent line, and the collision window is less than the time to transmit the header, a collision is guaranteed to never occur within the message text.

3.2.3 SCI-NET NETWORK LAYER

As is to be expected, the network layer of the ISO reference model is empty in the case of SCI-net. Since packets do not have to pass through intermediate nodes to reach their destinations this is an expected result and is characteristic of networks with fully distributed control.

It should be noted, however, that there is nothing to prevent a station to interface to more than one SCI-net link. In such instances it is conceivable, and indeed quite likely, that some form of communication across such networks would be desirable. This would require a store and forward mechanism at the intermediate nodes and modification of SCI-net software to support multiple virtual circuits between stations.

Such changes are not conceptually difficult and it is quite feasible to design such a store and forward

node with two (several) SCI-net interfaces, a processor, control ROM, scratchpad RAM, and some control circuitry. This is probably the best way to link several remote SCI-net installations since it is transparent to the basic SCI-net, intermediate nodes or gateways appearing as simple stations to any network they serve.

3.2.4 SCI-NET TRANSPORT LAYER

It has been noted that SCI-net software currently only guarantees no multiple receptions of messages when communication takes place with only one other station. A mechanism is provided whereby user tasks may request 'open channels' to be created restricting communications between the station making the request and the station which is requested. Of course, this does not restrict any other stations from using SCI-net except to the extent that communication between 'foreign' stations and those engaged in an open channel is blocked until the channel is released.

When open channel communications are to be terminated either station engaged in such communications may request that the channel be 'closed'. This is of course, a virtual circuit

protocol and SCI-net is immune to multiple open and close channel requests, thus correctly establishing such virtual circuits.

It should be noted that such a protocol need not be used and virtual circuits need not exist to support communications. SCI-net software merely provides facilities to handle multiple receptions of messages when communication takes place with one other station. If the application software can handle multiple receptions of packets then it may implement any degree of virtual circuit communication. SCI-net offers virtual circuits and multiple packet reception immunity under such circuits as a convenience to the application rather than a restriction.

3.2.5 HIGHER LEVEL SCI-NET FUNCTIONS

Currently SCI-net software offers the ability to send operating system level (FLEX) commands to remote stations. While an application may well be running on such remote stations to intercept such commands, SCI-net is capable of handling packets of such a nature itself. Although this may be construed as a superfluous feature of the network management software, it does serve to illustrate that although such features

have been implemented at the lower protocol levels, for the sake of efficiency, they do not in any way hinder user applications that have no need for them. In this sense virtual circuit support and the routing of packets to the operating system for interpretation are not layers built upon more primitive functions but are rather mini-applications of the network, illustrating primarily the correct, efficient, and cost effective nature of SCI-net.

3.4 LIMITATIONS OF SCI-NET

No description of a network or any other utility would be complete without commenting on the restrictions and limitations that exist. SCI-net is, of course, not the ultimate answer in local area networking and the environments in which it is not suitable must be defined.

When compared to systems such as Ethernet, one immediately comes to the conclusion that SCI-net is both slower and processor (software) dependent. This is quite true. In fact speed and hardware tradeoffs were the primary means of developing a very low cost network. Studies have shown [SHOC80] that networks linking small, single user systems exhibit low average

traffic and only marginally higher peak network traffic. Furthermore, the speed at which an individual station can consume messages from a network is severely limited by its processing power. In the case of microcomputer based systems the processing power is, as expected, quite limited. Even given efficient memory to memory data transfer facilities (DMA, not supported by SCI-net), the increase in effective communication rate is insignificant, the limiting factor being message consumption rate, a function of processing speed. Shoch and Hupp report [SHOC80] enlightening usage statistics on the Experimental Ethernet (2.94 megabit per second raw bit rate). Average utilization over a 24 hour day for 120 users was only 0.60% to 0.84%. Rates were higher during shorter periods but the busiest hour registered only 3.6%, and the busiest minute, 17%. Thus, demands on local processing power are not as severe as one might be led to believe.

It is relatively simple to show that software implementation of critical network functions such as carrier and collision detection, if done properly, impose little overhead on a single process system. If the network software is only required to examine the network upon receipt of incoming messages to the particular station in question, overhead is kept low since messages can not be sent faster than they can be

consumed. SCI-net approaches this ideal situation closely: only minimal overhead is required to determine that a message is not destined for a particular station, and message retries due to the inability of a station to consume the messages at the would-be data rate can be kept to a low rate (although too low a retry rate results in a reduction of effective data transfer rate and line utilization).

SCI-net requires the complete attention of the local processor when a packet is being transmitted and the corresponding acknowledgement is sent in response. Depending on the existence of virtual circuits blocking reception of the packet, this may take any length of time. However, if a single process is blocked pending successful transmission of a packet across the network then the time it remains blocked is inconsequential since it could not proceed until such a sucessful transmission.

The case of multiple processes running on a single station is more complex, for SCI-net will block <u>all</u> processes when only a single one has requested transmission of a packet. Despite the limited processing power of small microcomputers, some small operating systems do support such multiple processes, even in a single user environment, and SCI-net should

be able to interface with such systems. In its present implementation this is not possible without an apparent reduction in processing power corresponding to the transmission traffic. However, preliminary estimates and studies indicate that it may be possible to produce an intelligent SCI-net controller with the same complexity as the present one, at little if any, increased cost. With a local buffer or DMA access to main memory, the transmission process would be offloaded from the CPU and would therefore not slow the apparent speed of the system.

CHAPTER 4

SCI-NET HARDWARE

4.0 INTRODUCTION

The architecture of SCI-net has been developed over the first four layers of the ISO reference model. The network layer is empty, a direct consequence of the fully distributed control of the system.

This chapter describes the theory of operation of this SCI-net interface. Beginning with a block diagram, the circuit is described, section by section, with increasing degrees of detail and complexity.

4.1 - BLOCK DESCRIPTION

The SCI-net interface contains the following basic components: control circuitry to co-ordinate interface operation, an asynchronous communications interface adapter (ACIA) to convert parallel data to a serial

form, suitable for transmission on the network, a programmable interval timer to assist in keeping track of the busy/free status of the network, and, finally, the network interface. Figure 4-1 illustrates the main signal flow of the interface circuit in block form.

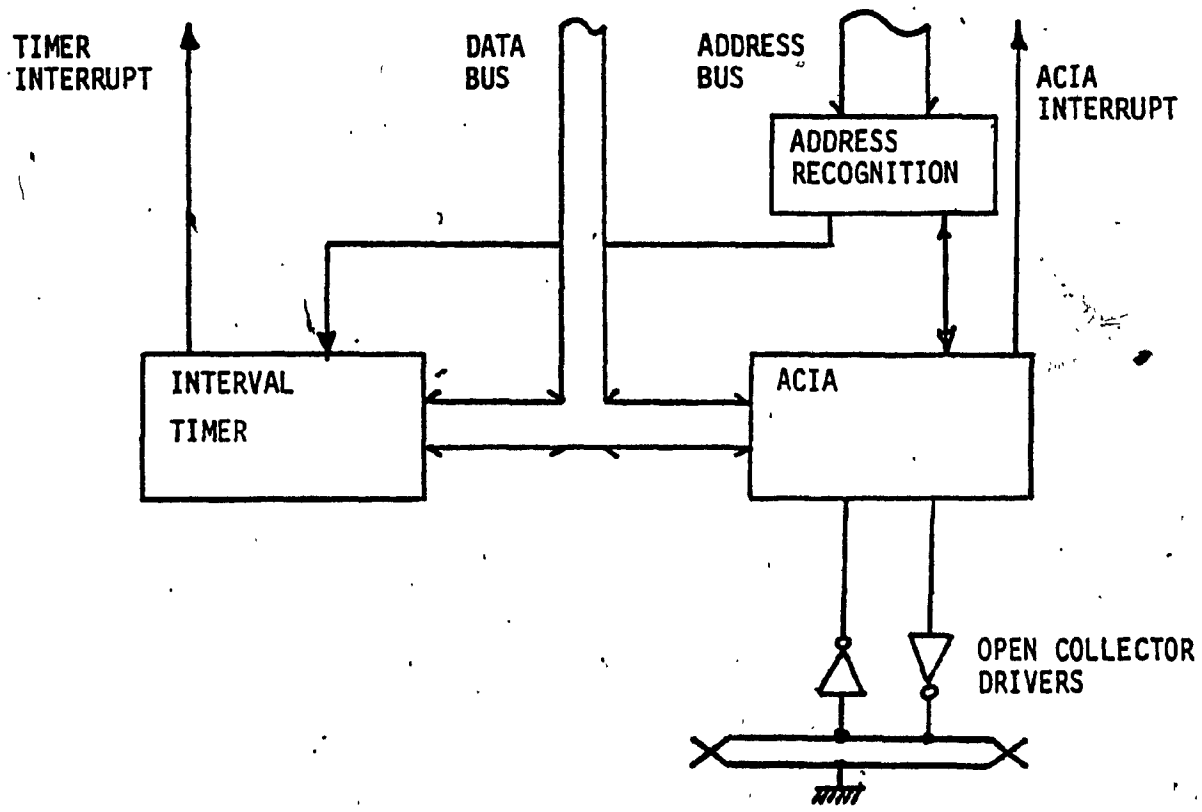

TIMER
INTERRUPT
DATA
BUS
ADDRESS
BUS
ACIA
INTERRUPT
ADDRESS
RECOGNITION
INTERVAL
TIMER
ACIA
OPEN COLLECTOR
DRIVERS

Figure 4-1

4.1.1 CONTROL CIRCUITRY

The control circuitry is responsible for determining when the central processing unit is communicating with the interface (i.e. when the interface is addressed by the CPU) and generating the clock signal that drives the ACIA and interval timer. When the proper levels are selected on the bus address lines the interface is selected and the processor may read or write to the various registers in the ACIA and interval timer. The ACIA clock is derived from the processor clocks. Two clock signals are available from the processor via the system bus. Commonly referred to as E and Q, these clocks usually run at either 1.000 Mhz or 2.000 Mhz depending upon the top speed of the processor and other system components. In the present experimental system they actually run at 0.895 Mhz rather than 1.000 Mhz due to the crystal in use on the central processor cards.

The SCI-net interface requires a clock rate of 2.000 Mhz for the ACIA. In 2.000 Mhz systems this is obtained directly from E. In 1.000 Mhz systems, advantage is taken of the fact that E and Q are in quadrature to derive a 2.000 Mhz clock from these

signals. Thus systems running at either standard clock rate may be present on the network.

Since the present experimental setup is running at somewhat less than 1.000 Mhz, the interface circuits cards in each station are configured for a 1.000 Mhz system clock and run less than top rated speed. This presents no problems so long as all stations on the network run at the same reduced speed.

4.1.2 ACIA CIRCIUTRY

The asynchronous interface adapter is a single integrated circuit that allows data transfer to and from the network. This device can be configured to operate at several data rates and data formats. The SCI-net interface ACIA is always configured to transmit and receive data at a bit rate that corresponds to one sixteenth the ACIA clock rate up to a maximum of 125,000 bits/second. Since the ACIA clock rate normally runs at 2.000 Mhz, the data rate is normally at the maximum data supported by this device.

This device has several registers that can be read and written by the microprocessor. These include a command register, a control register, a status

register, and a data register. The microprocessor writes the appropriate data to the command and control registers to set up the ACIA for the desired data rate and format. The status register can be read by the microprocessor (writing to this register results in a 'software reset' of the ACIA) to determine whether the ACIA can accept the next byte for transmission and if a byte has been received from the network. Note that the ACIA receives its serial input from the same line that its output drives. It is this that allows collision detection to be performed in software with little overhead.

4.1.3 INTERVAL TIMER CIRCUITRY

The interval timer, like the ACIA, is a single large scale integration (LSI) circuit. This device has three seperate countdown timers, each of which can be individually accessed by the microprocessor. The programmable interval timer (PIT) is accessed by the microprocessor through the use of several registers that control PIT timers and general operation.

Two of the three timers are used in the SCI-net interface. The first of these is driven by the same clock that drives the ACIA. Since the bit rate of the

ACIA is one sixteenth of this data rate, 16 ACIA clock cycles correspond to the time it takes to transmit (or receive) a single bit. For each byte transmitted there are ten bits sent: a start bit, eight data bits, and a stop bit. Thus the ACIA transmits a single byte in 160 ACIA clock cycles.

The least significant timer is setup to clock the next significant timer every 160 counts. Hence this timer produces a count for every byte transmission time. The PIT is used just for this purpose, to keep track of the probable end of a transmitted message. Thus network software can ignore the network when messages of no interest to a particular station are transmitted. Furthermore, the timer can interrupt the microprocessor when the station should monitor the network for incoming messages once again.

4.1.4 NETWORK INTERFACE CIRCUITRY

The ACIA interfaces to the physical network via two signals, the ACIA serial input signal, and the ACIA serial output signal. The serial output signal drives an open collector driver stage whose output is coupled to the network. The serial input connects directly to the output of this driver stage, and hence the network.

Except for a single gate propagation delay, the ACIA serial input follows the serial output exactly. At the present data rate, this propagation delay is of no consequence.

If the network were driven in a different fashion, then some signal conditioning would be required on the ACIA serial input. In the current experimental implementation of SCI-net, this is not necessary.

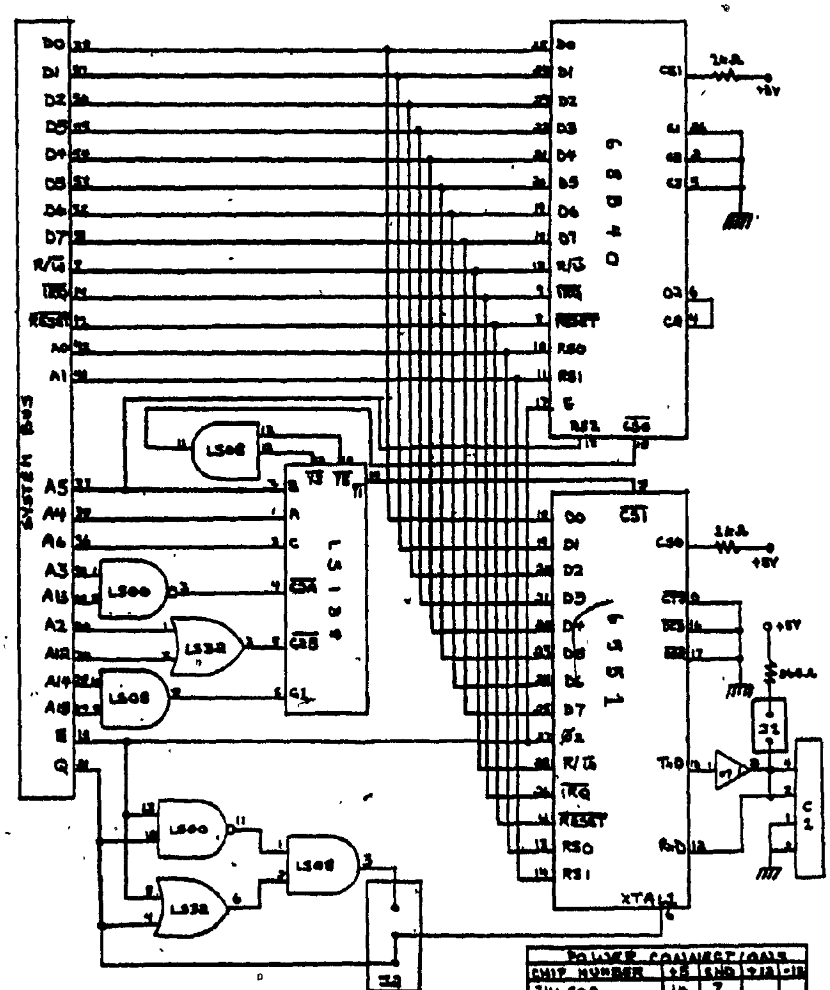
4.2 DETAILED SCI-NET INTERFACE DESCRIPTION

Figure 4-2 is a complete schematic of the SCI-LINK interface. This interface is in the form of a single circuit board (experimental prototype versions were wire-wrapped) that plugs into a slot of the bus of a station. The bus used is a proprietary bus, developed by Mr. J. Blaison of Concordia University. It supports many different eight and sixteen bit processors and is thus well suited for experimental work. The required bus signals for the SCI-net interface are available on any common microprocessor bus and thus designing SCI-net interfaces for standard busses presents no hardware problems.

As can be seen from Figure 3-2, the address

decoding circuitry is comprised of a 74LS138 address decoder integrated circuit and some support circuitry in the form of NAND and NOR gates. The ACIA occupies four memory locations addressed at $E018 (henceforth, a dollar sign before a value indicates a hexadecimal value). The PIT occupies two sets of four memory locations addressed at $E038 and $E058.

These addresses are not fully decoded. Thus the ACIA and PIT registers are duplicated throughout the memory map of the computer. This simplified the interface design of the prototype SCI-net interface somewhat and is consistent with the partial address decoding schemes used in other interfaces installed in the system.



POWER CONNECTIONS				
CHIP NUMBER	+5	GND	+12	-12
74LS00	14	7		
74LS08	14	7		
74LS32	14	7		
74LS138	14	7		
7407	14	7		
6551	15	1		
68040	14	1		

Figure 4-2

4.3 INTERFACE-SYSTEM SYNCHRONIZATION

The PIT is contained in the single Motorola MC6840 LSI circuit. As can be seen from the schematic, the same clock that drives the ACIA (Synertek 6551) also drives this device. It is important to note that although the timer specifications synchronize clock transitions to E, operation of a timer clock input above 1.000 Mhz is permitted provided that this clock is synchronized to E. This is indeed the case in this circuit. A separate, on-board clock would not only add to the component count, but would cause erratic operation of the circuit. The timer clock, and hence the ACIA clock, must be derived from E. In systems that are based on other processors, there is usually a system clock available for generating E and all derivatives thereof. However, in asynchronous bus systems, such clocks are usually not available and interfacing the SCI-net circuit to such systems may present problems. In such cases it would be necessary to use an on-card clock for the timer and ACIA and some means of communicating with the processor by means of the asynchronous bus protocol.

CHAPTER 5

SCI-NET SOFTWARE

5.0 SUPPORT SOFTWARE

The software to support SCI-net consists of several MC6809 assembly language routines that are resident in RAM along with the operating system when the system is running.

These routines have been optimized for speed and thus sacrifices have been made with respect to memory space and readability. However, they occupy approximately one and a half thousand bytes of memory and a set of entry point vectors simplifies routine linkage to application programs. Furthermore, these entry points are at fixed, predefined address to allow for compatibility between existing applications and future modifications to the network driver code.

5.1 SOFTWARE ROUTINES

The SCI-net driver is composed of routines designed to perform distinct functions. The sole purpose of some of these routines is merely to provide an elegant interface to external application code. The rest implement the actual handling of the SCI-net interface.

Interface routines provided include facilities to read a packet, write a packet, wait for a packet to become available and then read it, open and close secure channels (virtual circuits), and open a secure channel after waiting for any open channels to the current station to be closed by a remote station. In addition, there exists a routine for transmitting a special message. Special messages are described below.

5.1.1 SCI-NET READ CALL

The SCI-net READ routine transfers a packet in the read buffer to an application program's buffer. On entry the A register contains the length of the receiving buffer and the X register contains the address of the start of the buffer. On exit the A register contains the length of the message received, the B register contains the station identification of the source of the message, and the X register is

unchanged.

It should be noted that a packet must have been received when this routine is called otherwise the contents of the application program's buffer will be indeterminate. A status flag byte, described later, exists to allow applications to determine if a message has indeed been received.

5.1.2 SCI-NET READW CALL

This is a convenience call, waiting until a message has been received and then passing it to the application program. The entry and exit conditions are identical to the READ. Real-time programs should not make use of this routine since there is no limit to the time it takes for a message to be received. However, the majority of programs can freely use this routine since it relieves the program from checking for the arrival of a message.

5.1.3 SCI-NET WRITE CALL

The WRITE routine allows the transmission of a packet along the network. Since the time taken to

accomplish this function is not known, this call, like the READW call, should not be used in real-time applications. It locks the system into write mode until the transmission is finished and has been acknowledged.

Upon entry, the A register contains the length of the packet to send, the B register contains the station identification of the recipient, and the X register contains the address of the base of the packet to transmit.

In the present implementation the WRITE routine will attempt to transmit a packet indefinitely, there being no provision to abort after a specified number of attempts. Future implementations should allow a variable retry count.

5.1.4 SCI→NET OPEN CALL

The SCI-net open routine provides the mechanism by which a single virtual circuit may be established. Such circuits, their relationship to the SCI-net concept, and restrictions upon their use, are described in chapter 3.

On entry to the OPEN routine, the A register contains the station identification of the station with which a secure channel is requested. Since this routine calls WRITE, real-time programs should not make use of it.

This routine will exit when either a secure channel has been established, or a secure channel already exists with a station other than the one specified in the request. In the latter case, the A register will be set to zero upon exit.

5.1.5 SCI-NET OPENW CALL

This routine will repeatedly call OPEN until a secure channel has been established. As with the OPEN routine, the A register contains the station identification of the remote station with which a secure channel is desired. This routine is to be used when a currently open channel will be closed by a remote station. It removes the responsibility of constantly calling OPEN by the application program in such a case.

5.1.6 SCI-NET CLOSE CALL

This routine will close any secure channel currently open. Since it makes use of the WRITE call, it should not be used in real-time programs.

If no secure channel is currently open with the station in question then CLOSE will set the A register to zero on exit. If this call is to be used to check for this condition it is the responsibility of the application program to ensure that the A register is non-zero prior to calling CLOSE.

5.1.7 SCI-NET SPMESS CALL

Although not designed for use by application programs, this call is made available to the user should it be desired to implement additional protocol control functions in the enhancement of SCI-net.

SPMESS expects the same entry conditions as WRITE. It performs the same function as WRITE with one important exception: WRITE normally doubles DLE bytes encountered in the packet to be transmitted. This allows transparent transmission of eight bit data. SPMESS inhibits this doubling from taking place.

SPMESS must be entered with interrupts disabled

and it will exit with interrupts enabled unless the S_ACK bit (bit 3) within the internal SCI-net status byte was set. This bit it not normally available to application programs.

5.2 ADDITIONAL SCI-NET CALL INFORMATION

All SCI-net software routines are called as subroutines with either a BSR, LBSR, or JSR instruction, respecting the particular routine's entry requirements. All routines preserve all registers unless otherwise specified in the above description of the individual routines.

The entry point addresses of these routines are presently as follows:

READ	$2000
WRITE	$2003
OPEN	$2006
CLOSE	$2009
READW	$200C
SPMESS	$200F
OPENW	$2012

The dollar sign ($) indicates a hexadecimal quantity. This convention is adhered to throughout this thesis.

Note that currently the SCI-net routines reside in low memory on the MC6809 computer systems. While adequate for testing and some applications, when incorporated into the FLEX operating system, they should reside just below FLEX's base. This will require reassembly of the SCI-net drivers.

5.3 INTERNAL SCI-NET DRIVER ROUTINES

In addition to the SCI-net routines provided as an applications interface, there are several others that perform the actual protocol handling. Although these should not be used under any circumstances by application programs, they are documented here for the sake of completeness and to provide information for persons wishing to make custom changes and modifications.

It should be noted that, to discourage use by application programs, none of the following routines have entry points in the contiguous entry point block used for the application interface.

5.3.1 INTERNAL SCI-NET READ_I ROUTINE

This routine is entered whenever the ACIA on the SCI-net interface generates an interrupt. This routine accepts incoming messages and saves them in a buffer until they can be consumed. Details such as the handling of secure channels, message duplication when using virtual circuits, verification of properly formatted incoming packets, and the ignoring of messages not destined for a particular station are handled in READ_I.

After the initial ACIA interrupt that results in READ_I being entered, the ACIA is polled for each additional byte that is expected. This is necessary since a fully interrupt-driven scheme serving incoming data on a byte by byte basis would not be fast enough to receive an entire packet.

As previously described, when a packet header is received that is not destined for a particular station, interrupts from the ACIA are disabled and the programmable interval timer is set up to generate an interrupt when the message should end.

Although data is received from the ACIA on a polled basis, a timeout feature ensures that all receivers will not 'lock-up' should transmission of a packet cease unexpectedly.

5.3.2 INTERNAL SCI-NET RDBYTE ROUTINE

This routine performs the actual polling of the SCI-net ACIA for incoming data. The timeout function to prevent endless looping when a transmitter unexpectedly stops in the middle of a packet is implemented within RDBYTE.

5.3.3 INTERNAL SCI-NET TIMSER ROUTINE

TIMSER handles timer interrupts. The timer serves two purposes: first, it serves to implement some of the timeout functions within the WRITE routine such as checking for acknowledge timeout and retry after collision timeout; second, it keeps track of when ACIA interrupts are to be re-enabled after the transmission of a packet that is not destined for the particular station in question has ended.

TIMSER therefore either re-enables ACIA interrupts

or simply clears a flag depending upon the status of a flag in the SCI-net internal status register. This register is not available to application programs.

5.3.4 INTERNAL SCI-NET WRDRF ROUTINE

This routine is employed during collision detection. It waits for either the receive data register within the SCI-net ACIA to become filled with the character just received or timeout, indicating which of the two occurred with the A register either unaltered or zeroed upon exit.

Since collision detection in software is one of the unique aspects of SCI-net it will be discussed separately in section 5.6.

5.3.5 INTERNAL SCI-NET WRBYTE ROUTINE

This routine performs the actual transmission of a byte on the network.

5.3.6 INTERNAL SCI-NET DELAY ROUTINE

This routine implements a random delay. Such delays are used to ensure fair resolution of contention for the network after detection of a collision and during the initial request for access to the network. Presently, this delay is implemented by referring to a counter that runs between eight and sixteen byte times. Although it may appear that a uniformly distributed delay should be used with less chance of two stations using the same delay time, simulations indicate that there is little difference between the two methods.

5.3.7 INTERNAL SCI-NET LENGTH ROUTINE

This routine computes the true length of a buffer after DLE's are doubled.

5.4 SCI-NET STATUS REGISTERS

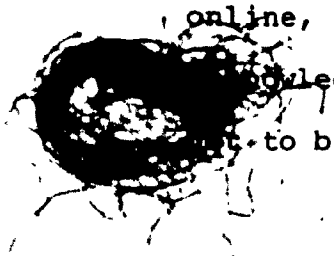
SCI-net software employs two status registers, one for internal use that is not available to application programs and another that is designed for use by application programs.

The status register available to application programs is a single byte currently present at address

$2015. The bits within this register are defined as follows: bit 0, the least significant bit, when set, indicates that there is a packet awaiting processing by an application program; bit 1, when set, indicates that an overrun has occurred, that is, a packet was sent to this station while the previous one was not yet consumed; bit 2, when set, indicates that a FLEX operating system command has been received; and bit 3, when set, indicates that FLEX is currently reading console input from the received FLEX command buffer. The other bits within this byte are, as of yet, unassigned.

Overrun errors pose no problem since acknowledgements are not sent for messages that cause overruns -- the messages will eventually be resent. Calling READ clears both bits 0 and 1. When a FLEX command is received (this is a type of special message) it is available to the application currently running, only being passed to FLEX upon termination of the current application. However, since it is available to application programs, a means is provided for them to recognize the intended destination of the command.

SCI-net internal status register bits, when set, indicate, from least significant bit on, that a remote transmitter is on-line, the local transmitter is



online, an acknowledgement is pending, an acknowledgement is being transmitted, and DLE bytes are to be doubled, respectively.

5.5 SCI-NET INTERRUPT INTERFACE

SCI-net requires patching into the interrupt service chain of whatever supervisor or operating system it is running with. Currently, it is presumed that no other devices other that SCI-net hardware are interrupt driven under the FLEX operating system. Since this certainly will not always be the case it is appropriate to describe the means by which SCI-net processes interrupts.

When SCI-net's interrupt handler is called, it determines the source of the interrupt, SCI-net ACIA or SCI-net timer. If it is neither of these the interrupt service ends with a RTI instruction. Clearly, if other devices are to be serviced on an interrupt-driven basis, then they must be checked for in the interrupt service chain. How this is done is not important as long as SCI-net ACIA receive data register full interrupts are dispatched to READ.I and timer interrupts are dispatched to TIMSER. It should be noted however, that delaying this dispatching too long

may result in incoming messages being lost and ACIA interrupts being re-enabled too late to receive the next packet. Thus it is recommeded that SCI-net interrupts be either given a high priority or, if this is not possible, increasing the number of preamble SYN bytes in a packet and/or programming the timer to interrupt early. The latter solution is discouraged since it implies increased overhead in SCI-net servicing and a somewhat reduced effective data rate.

It should be remembered that SCI-net send and receive software will tolerate additional interrupts of a higher priority than the SCI-net ACIA interrupts so long as the time required to service such interrupts does not hinder performance of the SCI-net software. Since the 6809 microprocessor does not support vectored or prioritized interrupts this is not a problem in the present implementation: all interrupts are masked while sending or receiving a packet.

5.6 SOFTWARE COLLISION DETECTION

At first glance it appears that software collision detection is a simple task: send a byte, receive it, and compare. Unfortunately, ACIAs perform double buffering of data. Assuming that the serial output is

connected to the serial input, a byte is transmitted and subsequently received in the following manner: the byte to be sent is placed in the transmit data register. It then progresses to the transmit shift register from whence it is serially shifted to the network. It is then shifted into the receive shift register. Soon thereafter, it is latched into the receive data register. This double buffering serves a useful purpose: it prevents gaps between sucessively transmitted bytes and thus ensures that the ACIA incurs no overhead in transmission.

When implementing collision detection however, it is desirable to defeat this double buffering. This can be accomplished in two ways: either the design of an ACIA offering direct access to shift registers or by ensuring that the correct byte is received and compared. Furthermore, an additional problem arises: since the effective elimination of double buffering results in gaps in transmission and thus a lower effective data rate, as few bytes as possible should be checked for collisions. SCI-net sucessfully limits this to a single byte.

Figures 5-1 through 5-13 show how collision detection is implemented on the transmitting station identification byte. Figure one represents the

situation when the SYN byte has progressed from the receive shift register to the receive data register. The receive data register full flag is turned on and this results in the send routine reading the SYN byte and discarding it, clearing the receive data register full flag as a side effect. As a result of the SYN byte being completely shifted from the transmit shift register to the receive shift register and consumed, the DLE byte is shifted from the transmit shift register to the receive shift register (Figures 5-1 and 5-2). After the SYN byte has been consumed, the transmit data register must be empty and the start of header byte (SOH) is placed in the transmit data register (Figure 5-3). When the DLE byte has been completely shifted to the receive shift register the SOH byte progresses to the transmit shift register (Figure 5-4). One bit time later, the DLE byte will have been transferred to the receive data register and the SOH byte will start to be shifted from the transmit shift register to the receive shift register (Figure 5-5). The receive data register full flag will be set. Consequently, the receive data register will be read and the flag cleared (Figure 5-6). The SOH byte will progress from transmit shift register to receive shift register to receive data register (Figures 5-6 to 5-8). It will then be consumed. At this point all the data registers within the ACIA will be empty (Figure 5-9).

The station identification byte is then placed into the transmit data register (Figure 5-10). It progresses to the transmit shift register, receive shift register and finally, the receive data register (Figures 5-10 to 5-13). It is then read and compared with the identification byte sent. If they are not the same, then a collision has occurred.

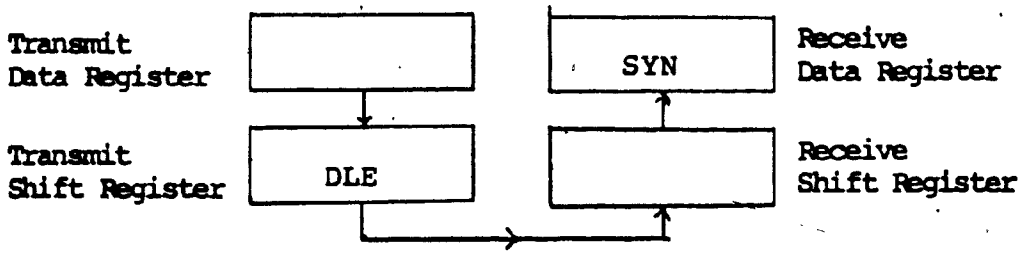


Figure 5-1

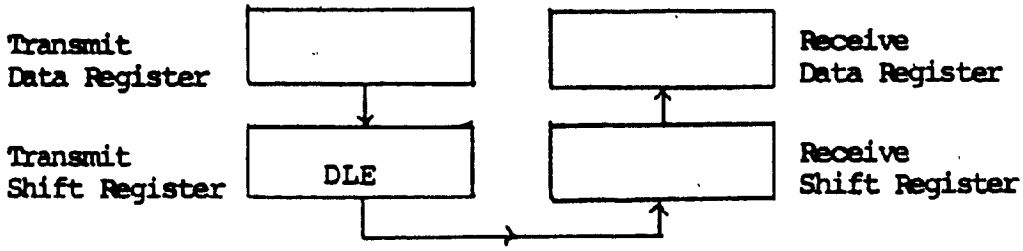


Figure 5-2

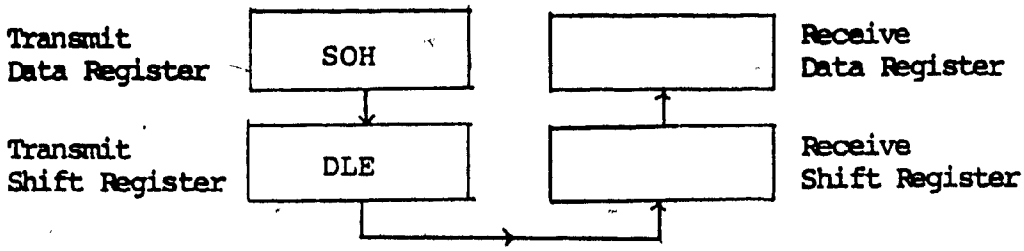


Figure 5-3

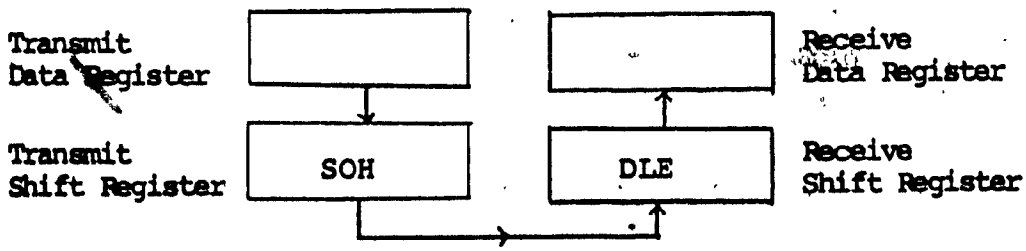


Figure 5-4

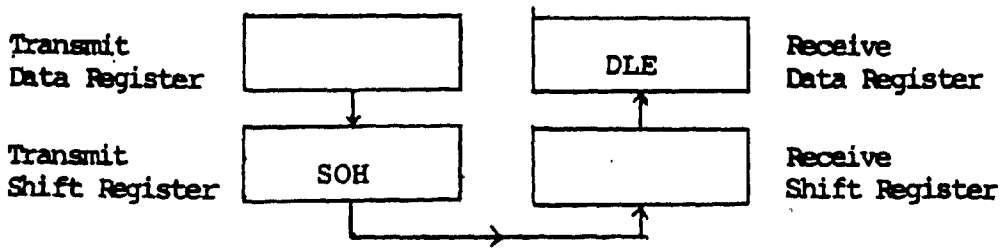


Figure 5-5

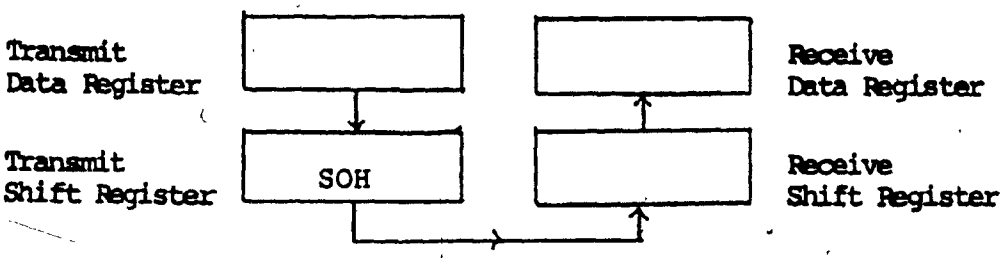


Figure 5-6

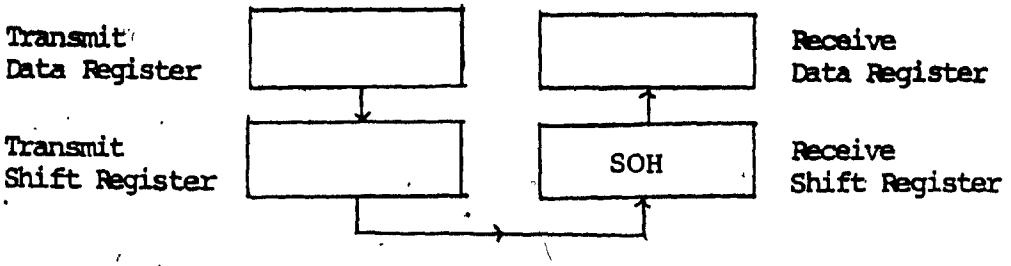


Figure 5-7

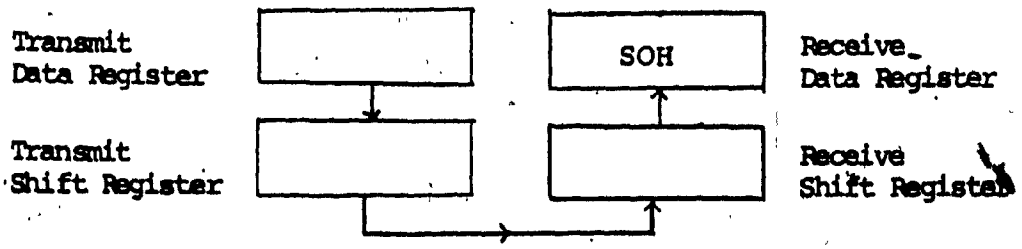


Figure 5-8

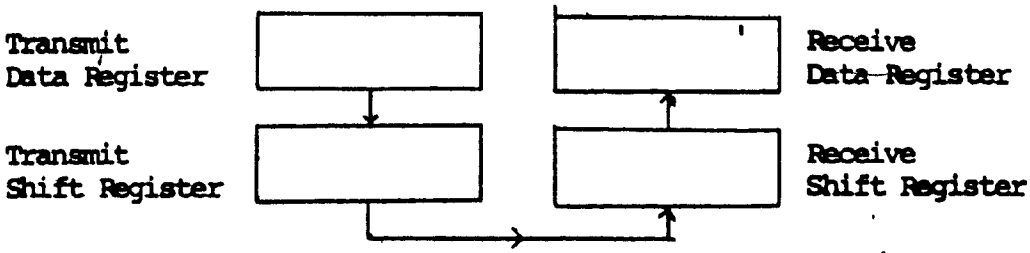


Figure 5-9

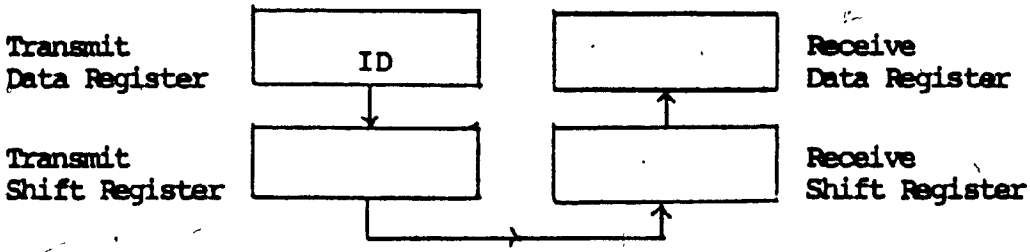


Figure 5-10

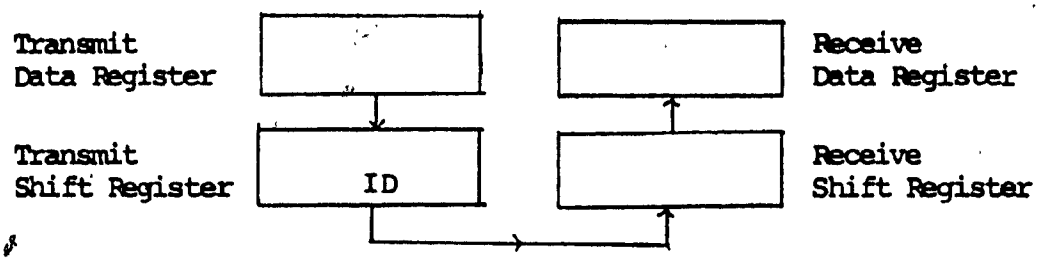


Figure 5-11

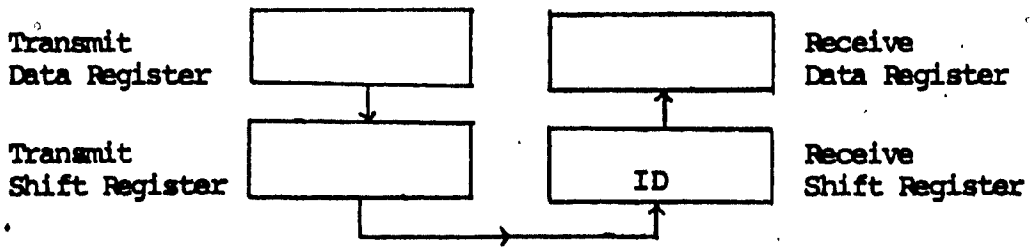


Figure 5-12

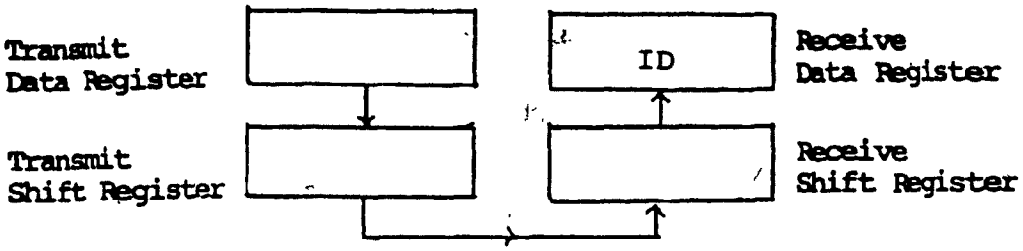


Figure 5-13

There is an additional problem, however. If a collision occurs then the ACIA may never indicate receive data register full. Furthermore, no error flag bits may be set either. This phenomenon was observed while testing the network software. A hard coded timeout was implemented to detect this and ascertain the presence of a collision. If this were not done, SCI-net software would wait indefinitely for receipt of a byte that will never arrive. Of course, the next transmission that occurs will clear this loop but two or more stations will be inoperative until this occurs, clearly a circumstance that is to be avoided.

CHAPTER 6

TESTING AND SIMULATION

6.0 INTRODUCTION

To adequately determine whether a network performs with a good line utilization, efficiency, and to aid in fine tuning such a network, numerous measurements must be made on the network in operation. Testing is an integral part of the design of such a system since it allows the designer to determine operating parameters and limits.

Often, there are situations where testing is either difficult, expensive, or impractical to perform. With respect to SCI-net, the network can only be tested with four stations, the limiting factors being the number of SCI-net interfaces available, and, more importantly, the number of stations available in our laboratories. A computer simulation was therefore used to investigate configurations that could not be practically tested with the present equipment. While

simulation is no substitute for actual real-time measurements, it offers the advantages of being able to gather more data at extreme limits of operation, and can help to dictate the steps required to fine tune a system. This chapter is concerned with the testing and simulation of SCI-net with several operating parameters varied.

6.1 SIMULATION WITH GPSS

GPSS is an abbreviation for General Purpose Simulation System. Originally developed for IBM 360 and 370 computer systems, early versions such as GPSS III and GPSS V were quickly adapted for other machines.

GPSS handles constructs such as blocks, entities, and transactions. A transaction is an abstract construct that moves from block to block within a system being simulated. Consider a simple example: Parts are machined in a machine shop. The parts are transactions and the various steps that they undergo in being machined are blocks. A part arrives to be processed. It must wait for a machine and is thus placed in a queue. When a machine is available, the part is machined, leaving the queue. When machining is finished, the part leaves the system.

In this example, the action of arriving, being queued for a machine, obtaining a machine, being machined, releasing a machine, and leaving are all blocks that a transaction goes through. An entity may be a queue, storage, facility, chain, etc. Queues carry their usual connotation and thus need not be explained. A storage is an entity that may hold no more than a fixed number of items. A facility is a resource (such as a machine, to use the previous example) that can be used by only a single transaction at a time. Note that if any part could be machined on any one of three machines, a storage of capacity three could be used to represent the machines. Chains are a special type of queue. Items may be removed in any order and any number of items may be removed at once. An application of chains is the implementation of 'balking'. Consider people entering a queue. After a certain time in the queue they decide not to continue to wait to be processed and decide to leave. They are said to have balked.

In the GPSS program used to simulate SCI-net, the timeout function while awaiting an acknowledgement is implemented in this way: a transaction representing a packet just sent is split into three and the three copies are dispatched to different blocks. One enters a chain indefinitely. A second is transformed into an

acknowledgement packet and waits until it can be transmitted. The last copy enters an advance block where it waits for a fixed time -- the timeout limit. When the acknowledgement is successfully transmitted, it removes the original transaction from the chain and sends it to a block that counts it as a packet received by the intended receiver. The acknowledgement transaction is destroyed. Should the timeout expire, the third copy will remove the original transaction from the chain and dispatch it to retry. Again, the copy will be destroyed. Removing non-existent transactions from a chain simply performs no function.

At this point the reader should realize that GPSS programs are non-procedural and that there may be many seperate transactions at a particular block. GPSS normally allows a transaction to proceed as far as it can before processing other transactions that occur simultaneously. However, this ordering may be changed. Furthermore, the means by which a transaction can match another one on a chain is simple to implement in GPSS, and the reader should not concern himself with the details. A GPSS program can be considered analogous to a re-entrant segment of code within an operating system: many processes, or transactions can be executing a particular instruction unless restricted by semaphores (facilities).

GPSS supports the existence of non-sharable data storage locations, or 'savevalues'. In addition, transactions may have several parameters associated with them. Thus, if a transaction represents a packet, its parameters determine its attribute, i.e. whether it is an acknowledgement or not, the source of the packet, its destination, etc. In addition, when random timeouts are to be simulated, the distribution of the random variable can be strictly defined. In short, GPSS provides a flexible tool for simulating systems of transactions.

6.2 SIMULATING SCI-NET

A GPSS program, listed in Appendix B, was used to simulate the SCI-net system. GENERATE blocks produce packets that are to be transmitted and acknowledged. It is ensured, by means of GATE and LOGIC blocks, that only a single packet is to be transmitted by a particular station at any given time. GENERATE statements normally produce transactions as dictated by a random distribution function, and the SCI-net model is no exception: packets arrive for transmission with an exponential distribution with a mean of one byte time. However, they do not arrive, under any circumstances, if a transmission by the particular

station is in progress. This properly models a single process system where network servicing preempts other tasks.

Once ready to be transmitted, a packet or, more correctly, the GPSS transaction that represents a packet, checks if the line is currently being used by anyone. If not, it proceeds to mark that it is using the line and transmission begins. Clearly this is a semaphore, since both operations, testing the bus and starting to transmit, occur at the same time. However, since, in reality, this is not an indivisible operation, the GPSS model incorporates a delay in an ADVANCE block, that corresponds to the collision window. Other ADVANCE blocks are used to mark random delays, header transmission times, and message text transmission times.

After the collision window delay, transmission of the header is simulated with an ADVANCE 60 block. Once a packet transaction has left this block it tests whether, in fact, it indeed was the only one transmitting. If so, transmission of the rest of the message continues in an ADVANCE FN$LEN block. The time spent in this block is determined by a function, LEN, that returns the transmission time as a function of current simulation packet length and the type of

message (acknowledge or normal).

If a collision does occur, a savevalue (COLL) is incremented, and the packet waits for the line to clear. When the last packet involved in the collision detects the collision, COLL will equal a count of the number of stations that attempted to transmit simultaneously, kept in save value BUSRQ. At this point all transmitting stations will have detected the collision, BUSRQ and COLL will be set to zero, and attempts at transmission continue in a normal manner. Collisions and their resolution are thus simulated accurately in the GPSS program.

Once a packet has been transmitted, a test is made to check if it was an acknowledge packet. If not, the the sender must await receipt of an acknowledgement. The original packet transaction is split, with SPLIT blocks, into three. The first is linked to a timeout chain, the second proceeds to an ADVANCE block where the wait for acknowledge timeout is implemented, and the third has its parameters changed so that it effectively becomes the acknowledgement. Simulation of the transmission of this packet then proceeds. Should the timeout expire, the original packet transaction will be unlinked from the timeout chain, and an attempt will be made at retransmission. When the acknowledging

packet is finally transmitted, it unlinks the waiting packet transaction and causes it to be sent to be destroyed and counted in a TERMINATE block.

As was previously mentioned, the unlinking of a transaction that does not exist on the chain specified has no effect, so the first transaction that causes unlinking to take place determines the destination of the unlinked packet. This is a standard method of simulating balking behaviour in GPSS programs.

The simulation of the network also takes into account the inability of an acknowledgement to be sent when the acknowledging station is attempting another transmission (this could not occur if the original packet was not received), and other sundry details.

By running the simulation for several different packet lengths, wait for acknowledge times, and different numbers of transmitting stations, statistics related to line utilization can be collected.

6.3 SIMULATION RESULTS

The GPSS program was run with two, four, and eight stations transmitting to an identical number of

receivers. Measurements of line utilization under saturated conditions were taken for packet text lengths of 32, 64, 96, 128, 160, 192, 224, and 250 bytes. Furthermore, the simulations were run with wait for acknowledge times of 250, 500, 1000, and 2000 byte times. The line utilization results are tabulated in three graphs, plotting utilization as a percentage against packet length, with four curves, indicating the four wait for acknowledge times. These graphs are presented in Figures 6-1 through 6-3.

Examining these graphs leads to several observations. First, increasing the wait for acknowledge time can result in dramatic improvements in line utilization, especially when many stations are on the network. The reason for this is straightforward: With many stations attempting to transmit at the same time, acknowledgements can be significantly delayed. As a result, if a short timeout is in effect, the station waiting for the acknowledgement will attempt to resend, to no avail, since the receiver is busy attempting to transmit its original acknowledgement! The wait for acknowledge timeout is again entered and the result is devastating: collisions between competing stations increase sharply, and performance degrades rapidly. Increased packet length compounds the problem since the time before a station can contend for the

network when another station has control of it is also increased.

One cannot increase the wait for acknowledge time indefinitely. There are situations where messages will be lost and an unnecessarily long wait for acknowledge time will increase the time required to detect the lost messages. In reality, when a wait for acknowledge timeout occurs, there exists the possibility that the mesasge was not lost, but rather the acknowledgement could not be sent in time. A wait for acknowledgement time should thus be chosen carefully. Simulations however, indicate that choosing this operating parameter to be too short is far worse than making it too long.

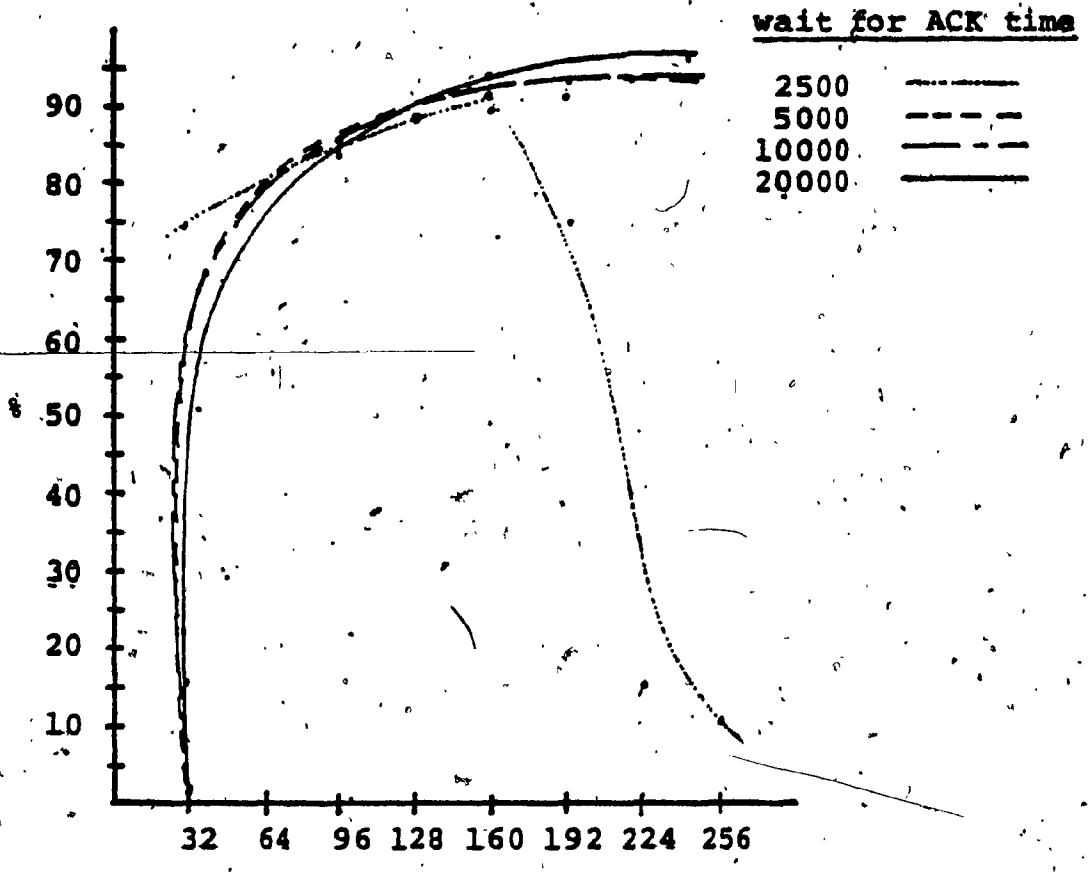
It should be noted that the simulations dedicate a single receiver for each transmitting station. In practice this is not so, and thus there are greater numbers of lost messages. However, if virtual circuits are used to establish long communication paths, these lost messages will only be retransmitted whenever a wait for acknowledge timeout occurs. This is another reason in favour of a long timeout limit over an excessively short one.

An interesting phenomenon, observed in actual

testing, but not adequately explainable until simulations were run, is the local peaking of the line utilization curve at certain packet lengths. A theory was developed during testing to account for this behaviour: a resonance condition exists whereby a station transmits, another station transmits, the first transmission is acknowledged, the second transmission is acknowledged, and the process continues. Although the peak is dependant on random delay and timeout functions, certain combinations of packet length, wait for acknowledge times, and number of transmitting stations can cause this condition to occur. It appears that once resonance occurs, stations tend to stay locked in resonance until sufficiently disrupted by other transmissions. With larger numbers of stations the effect is, of course, not as strong as with smaller numbers of stations. Furthermore, under normal operating conditions, such resonance will arise only rarely, and thus need not be considered when setting operating parameters for a particular SCI-net installation.

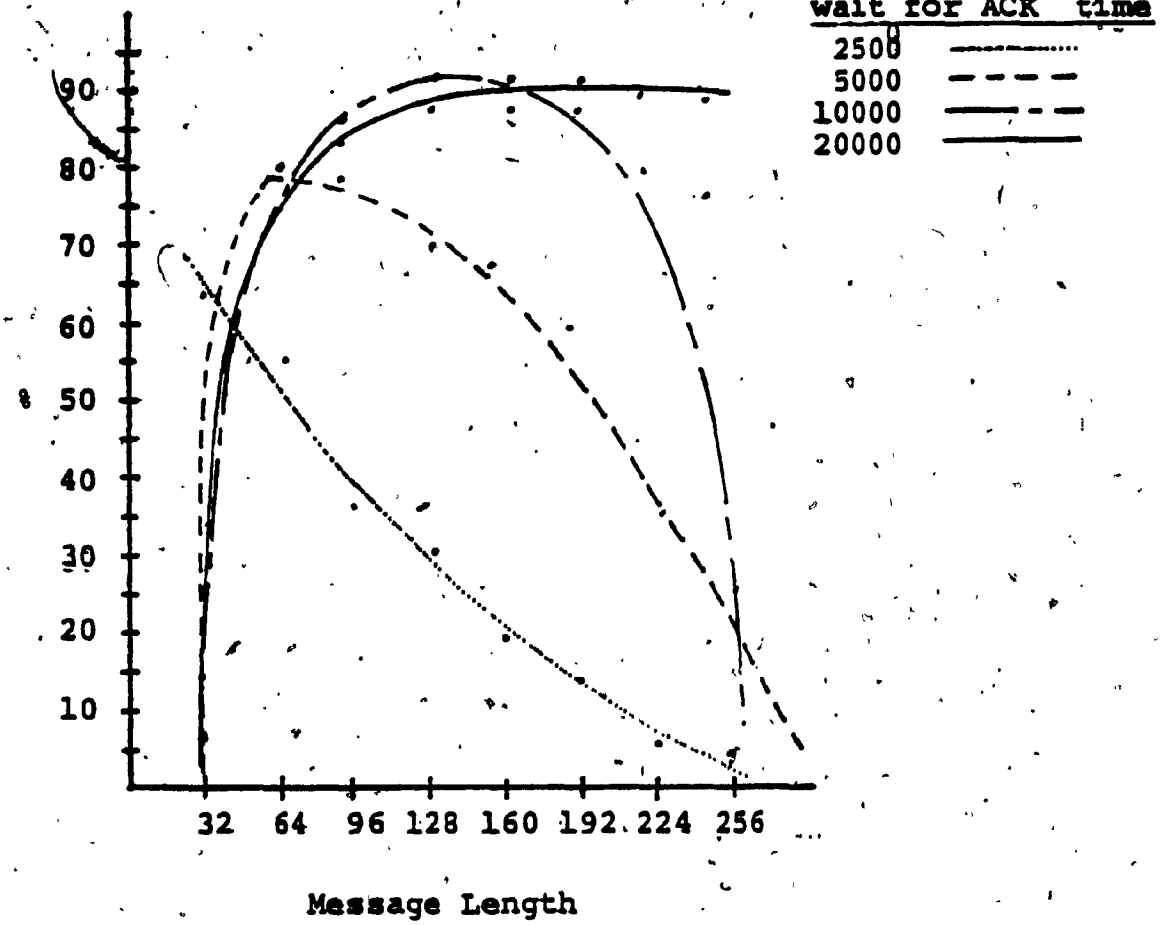
In addition, a simulation was run adjusting the collision windows and collision detection time to make the system correspond to an Ethernet-type network. Note however, that the same network protocol is used. Ethernet's network protocol is significantly different.

Therefore the results can only be used to compare hardware implementation of carrier sensing and collision detection with software sensing of the same, and not Ethernet with SCI-net. The resulting graph is plotted in Figure 6-4.



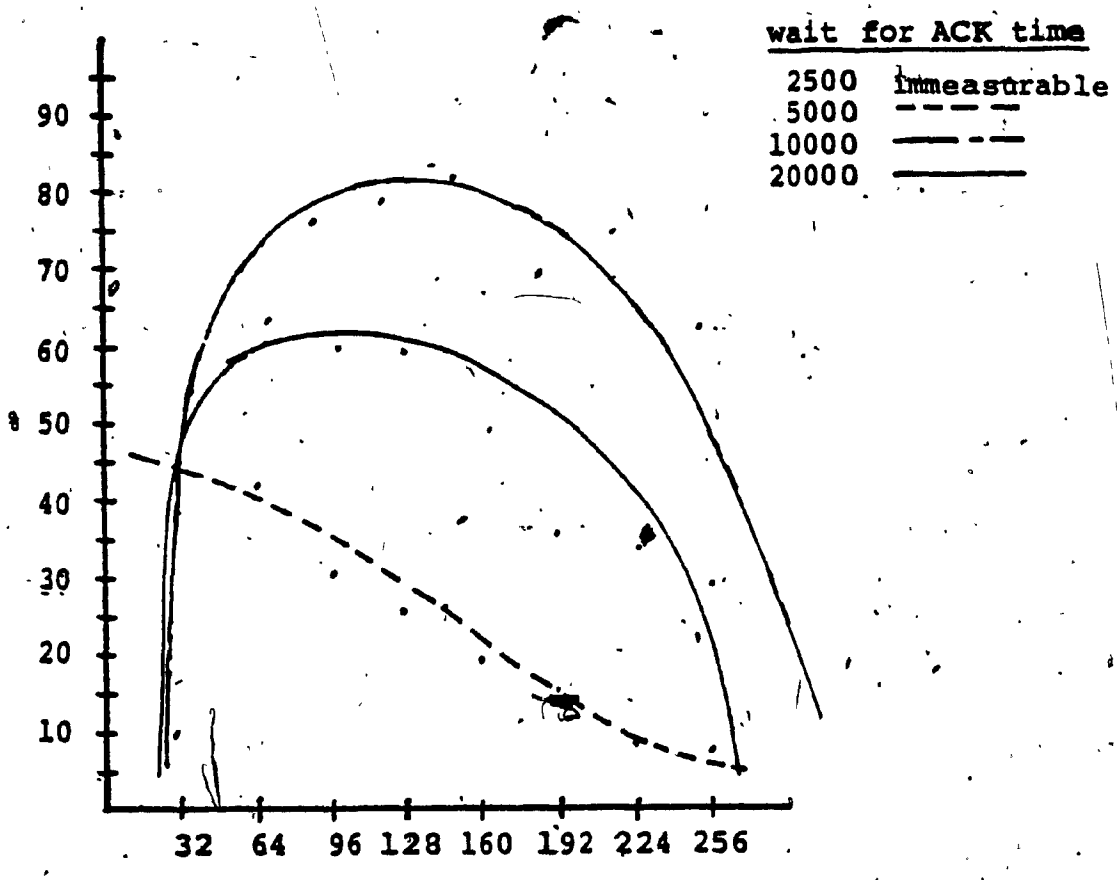
%-utilization for 2 stations

Figure 6-1



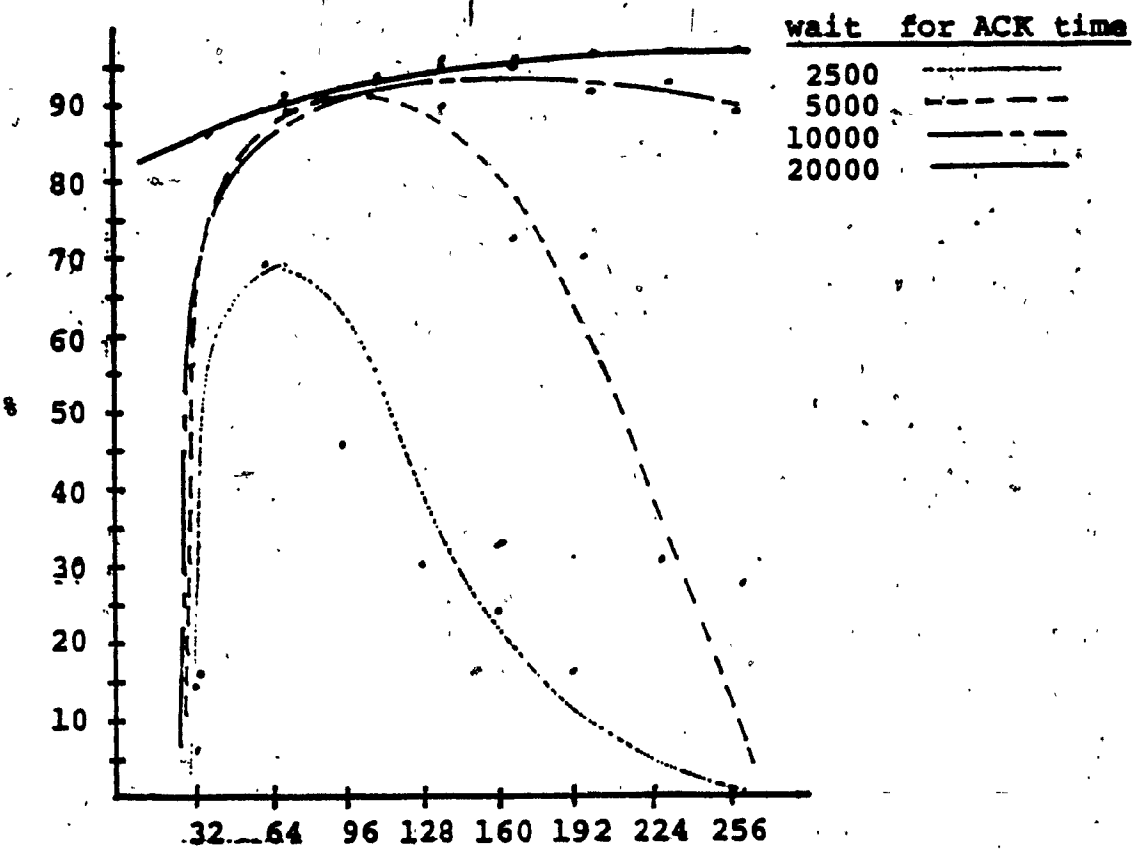
%-utilization for 4 stations

Figure 6-2



%-utilization for 8 stations

Figure 6-3



hardware-based system
%-utilization for 4 stations

Figure 6-4

6.4 OBSERVED MEASUREMENTS

In addition to running simulations, actual tests were made with two stations transmitting to two others. The results are tabulated in Figure 6-5. The results agree closely with the simulation of the network under the same conditions, confirming the validity of the simulation model. Any discrepancies can be accounted for by the simple fact that although the simulations are accurate, they are not perfect. For example, it would not be an easy task to simulate the timing of the MC6809 microprocessor as it processes instructions in the SCI-net software. Such a simulation would be extremely complex and a reasonable limit must be chosen as to how closely it is desired to mimic the behaviour of a system with computer simulations.

Msg. Length	Station Transmissions Monitored					
	A TO B			C TO D		
	Sent	Coll.	Lost	Sent	Coll.	Lost
32	10971	1495	356	11447	1597	257
64	8355	1074	316	8588	1202	242
96	4258	303	112	4279	339	97
128	5523	566	210	5556	580	192
192	4028	171	107	4014	199	79

Figure 6-5

6.5 NETWORK OVERHEAD

A discussion of SCI-net would not be complete without an analysis of the overhead that servicing the network places on each local computer. It has already been made clear that when a station is executing the network driver routines, either to send a message or receive an incoming message, all of the processing power is dedicated to this task.

The size of this fraction for a local processor is, of course, proportional to its utilization of the data link for sending and receiving messages, and is therefore highly dependent on the type of job being performed by the local user. The overhead may, however, be characterized by an estimate of the time required for the three distinct functions performed by the interface software: line monitoring, message reception, and message transmission.

6.5.1 LINE MONITORING

For any line activity all local computers receive the message headers. The overhead of this function is dependent on the header length. Given that the header length is h bytes, the message length is m bytes, and

the time required to perform any software operations not directly related to data reception (responding to initial ACIA interrupts, setting up the interval timer, etc.) is x, expressed in byte times, then overhead during reception of packets not destined for a particular station is given by

$$\frac{h + x}{h + m} .$$

In the present implementation, this is equal to

$$\frac{10 + 0.25}{10 + 256} = 0.0385 .$$

This figure presumes a message length of 256 bytes. For a 100% saturated system, this would take 3.85% of the processor time. Line utilization under non-saturated conditions is directly proportional to the number of users and inversely proportional to the theoretical data rate. Scaling the results of the observations by Shoch and Hupp [SHOC80] to a SCI-net system supporting 24 users yields a figure of less than 5% average line utilization. Thus the expected overhead due to line monitoring is

$$0.0395 \times 5\% = 0.197\% .$$

6.5.2 MESSAGE RECEPTION

In this case, the maximum processing overhead is 100% while a message destined for a particular station is being received. Thus the component of overhead attributed to message reception is directly proportional to the time spent receiving messages for the station in question.

Assuming that message transmissions are equally distributed among all stations, then, given 24 stations on the network, this figure becomes

$$1/24 \times 5\% = 0.21\%$$

for the 5% average line utilization postulated above.

The preceeding discussion has not considered the time required to acknowledge a packet transmission. This depends greatly upon network traffic and should be treated in the same way as message transmission, covered in section 6.5.3. Furthermore, sections 6.5.1 and 6.5.2 presume that no overhead is incurred during the reception of a header destined for a particular station and that 100% overhead is incurred when a packet destined for a particular station is received.

The reader can also consider the overhead of message reception as being somewhat less than 100% and that all headers require processing overhead. The only consequence is a redistribution of overhead between header and message reception. Since these are disjoint events, the total value remains the same.

6.5.3 MESSAGE TRANSMISSION

The overhead due to message transmission is that fraction of time that a station spends transmitting. This is not only dependent upon the volume of data that a station must transmit but also on the time required to transmit any given packet. This includes the time to wait for a free network, transmit partial message headers until no collision is detected, transmit transmit the remainder of the header, transmit the message, wait for an acknowledgement, and, if the acknowledgement is not received within the specified delay time, repeat the entire procedure. During this interval, the station is not available for any other processing.

Assuming c collisions, and t acknowledgement timeouts, the time for a packet to be sent is

$$c(t_h + t_c) + t(t_p + t_a) + t_p + t_r$$

where t_h is the time to transmit the header fragment before collision detection is performed, t_c is the collision back-off time, t_r is the acknowledgement response time (less than t_a), t_p is the packet transmission time, and t_a is the wait for acknowledgement time. Since t_c is a random variable, the value of its mean may be used in the above equation to determine an expected time for the transmission and subsequent acknowledgement of a packet. Since c and t depend upon line saturation conditions, only an estimate of their values can be made in the above equation. However, the number of collisions per transmission and the number of acknowledgement time-outs will increase with increased line utilization. Observations indicate (Figure 6-5) that, with two transmitters sending to two receivers, under saturated conditions, the mean value of c is less than 1/5 for all message lengths. With large numbers of stations, this may increase due to the necessity of resolving n-way collisions. The mean value of t is typically less than that observed for c, although, as has been indicated by simulations, this value can increase sharply with increased message length if t_a is not kept large enough.

CHAPTER 7

CONCLUSIONS

7.0 SCI-NET's PLACE AMONG NETWORKS

The concept of a network with almost complete control of protocol functions in software has been demonstrated to be a viable one. Experimental testing with four stations indicates that line utilizations in excess of eighty percent can be expected. Furthermore, simulations indicate that SCI-net can support up to two dozen single-user small micro-computer systems in an office or laboratory environment with acceptable efficiency. Thus the primary design objectives of developing a very low cost CSMA/CD network have been met. Four experimental network interfaces were assembled at a total cost of approximately one hundred dollars. The goal of the work was to design, implement, and demonstrate the concept of a software controlled CSMA/CD network. This has been achieved, and the feasability of software control of a CSMA/CD network has been demonstrated. The results to date are

very encouraging and indicate that research should continue in the area of such networks.

Besides the clear advantages that SCI-net offers, one must not overlook the limitations. Some of these are due to the simple fact that the primary goal of the work was to demonstrate the feasability of a software controlled CSMA/CD network. Optimal control of such a network was a secondary concern. SCI-net can clearly benefit from better designed control software, multiple buffering and virtual circuit support, and improved matching of transmitter data retry rate to receiver data consumption rate. Since the basic design has been shown to be sound, the steps to be taken to effect these improvements involve enhancements and upgrades to the SCI-net software, retaining, of course, the collision detection algorithms and the basic protocol. The rest of SCI-net's limitations are, however, design dependent. SCI-net requires some local processing power. While this is not significant in a single task environment, the pre-emption of all other services while attending to the network would result in pre-emption of task scheduling in a multiple task environment. The end result would be that tasks that otherwise could be scheduled to run could not due to SCI-net overhead. This can be viewed in an interesting way: if the average overhead were to be distributed

over all processes, then the effect would be similar to that of processor slow down. While this would not prevent such a system from working, it may be undesirable. Real-time programs can be supported by SCI-net as long as the transmit priority remains above that of the real time task input. Lowering it creates the risk of a broken transmission sequence. This can be solved in the case of real time tasks with an interrupt interval longer than that of the transmit function, but higher frequency tasks would require hardware modification.

In a single process, small microcomputer-based system, SCI-net works very well, since the typical volume of data flow across the network has been shown to be within the limits that SCI-net dictates. The slow down due to multiple processes would necessarily limit demands on the network and would probably cause them to be within design limits as well. However, large minicomputer and mainframe systems require data transfer bandwidths far in excess of that which SCI-net is capable of. The higher cost of traditional networks such as Ethernet, DECnet, and others are well justified for such applications.

It appears then, that the biggest obstacle to SCI-net type networks is the demand on the host

processor. If this could be removed, then small, multiple process systems could probably be supported. As software and hardware sophistication increases, such systems will become available in price ranges that warrant the use of low cost networks such as SCI-net.

One method of removing the burden of supporting SCI-net from the host processor would involve the development of an intelligent SCI-net interface. Currently there is a microprocessor available, the Motorola MC6803 that offers an instruction set very similar to the MC6809 used in the SCI-net experimental network. It is based on the Motorola MC6800, the predecessor to the MC6809, but exhibits better instruction times than the MC6800. This processor includes an on-chip ACIA, interval timer, and some RAM. Although restrictions apply to the simultaneous use of the ACIA and timer, a feature is included that can be used to automatically detect the end of a transmission: the ACIA can be programmed to ignore incoming data until ten consecutive mark (logic 1) bits are detected. This feature eliminates the need for a timer in the critical message timeout state. As a result, an intelligent SCI-net interface could be implemented using an MC6803, an external program ROM, some RAM, a network driver and host bus interface. Proper design could conceivably keep the component count under a

dozen devices and result in an interface only marginally more expensive than the present unintelligent one. Furthermore, the Motorola MC6801 is functionally identical to the MC6803 with the exception that it offers two kilobytes of mask programmable read only memory on-chip. In large quantities, the MC6801 is available at extremely attractive prices.

At the time of this writing at least one company has made use of the MC6801 family of devices to implement a network: Coleco has recently introduced their 'Adam' personal computer. This device utilizes several MC6801-type microprocessors to service the printer, keyboard, and external devices. Coleco promotional literature indicates that plans for an 'Adamnet' to network several such computers are being formulated.

This system, however, requires a MC6801-type processor to act as a central controller and performs network arbitration in hardware, thus not implementing a CSMA/CD network with fully distributed control.

Proponents of hardware based systems argue that they are faster than corresponding software based systems and that cost is of no concern since sufficient demand will result in the production of VLSI (very

large scale integration) devices to replace up to several hundred individual devices. While this is certainly true, the cost of correcting hardware faults and fine tuning such a system to meet particular environmental needs is exorbitant. In addition, the inflexibility of hardware based systems results in a large time factor in the development of upwardly compatible enhancements to such systems. The cost of making changes to software is several orders of magnitude less and field changes after delivery of equipment can be implemented with relative ease.

To draw an analogy, consider microprogrammed computers. Although slower than their hardwired counterparts, they are nevertheless in great demand. Those with writable control store can, in fact, be field tailored for particular applications. One must always be careful, however, not to propagate a multitude of incompatible versions of the same thing.

Software emulation of traditionally hardware oriented systems has become an attractive option for the aforementioned reasons and more; it is merely a logical progression to apply this concept to networking as well.

REFERENCES

[AGRA81] A. K. Agrawal, V. V. Vadakan, "Jet Propulsion Local Area Network (JPLAN)", Proceedings of the 2nd International Conference on Distributed Computing Systems, 1981, pages 360-368.

[BENH81] Eric Benhamu, Zilog, Inc., "Integrated Software Design for Z-Net, a Local Microcomputer Network", Proceedings of the 2nd International Conference on Distributed Computing Systems, 1981 pages 397-403.

[BERN81] G. Bernard, "A Non-Persistant CSMA-Abort Protocol for a Local Computer Network", Proceedings of the 2nd International Conference on Distributed Computing Systems, 1981, pages 369-376.

[BIRZ81] P. Birzele, H. Thinschmidt, "A Local Distributed Microcomputer Network Based on an Optical Bus System with Decentralized Communication Control", Proceedings of the 2nd International Conference on Distributed Computing Systems, 1981, pages 497-502.

[BUHR82] R. J. A. Buhr, S. Michell, "Object-Oriented Structured Design of Layered Protocol Systems", Proceedings of the 3rd International Conference on Distributed Computing Systems, 1982, pages 288-293.

[CROU82] C. W. Crouch, "AOS - A Tool for Designing Distributed Real-Time Operating Systems", Proceedings of the 3rd International Conference on Distributed Computing Systems, 1982, pages 422-429.

[FRAN82] W. R. Franta, J. R. Heath, "Performance of the Hyperchannel Networks: Parameters Measurements, Models and Analysis", University of Minnesota Technical Report 82-3", January 1982.

[GADS82] J. A. Gadsden, "ADNET: An Experiment in Computer Networking for the Royal Navy", Proceedings of the 3rd International Conference on Distributed Computing Systems,

1982, pages 351-357.

[HUTC82] D. Hutchisson, P. Cocroran, "A Microprocessor-Based Local Network Access Unit", "Mircoprocessors and Microsystems, Vol. 6, No. 1, January-February 1982, pages 3-8.

[ISO] ISO document ISO/TC97/SC16 N719.

[KNIG81] J. Knight, M. Itzkowitz, "THC - A Simple High-Performance Local Network", Proceedings of the 2nd International Conference on Distributed Computing Systems, 1981, pages 354-359.

[LEVE82] S. Leventis, G. Papadopoulos, "Further Simulation Results on The Performance of a New Double-Loop Computer Network", Proceedings of the 3rd International Conference on Distributed Computing Systems, 1982, pages 467-473.

[PAUL82] M. Paulk, "Some Comparative Measurements of Computer Interconnection", Proceedings of the 3rd International Conference on Distributed Computing Systems, 1982, pages 456-460.

[SELI81] D. R. Seligman, "On the Performance Evaluation of DECnet", Proceedings of the 2nd International Conference on Distributed Processing, pages 476-483.

[SHOC80] J. F. Shoch, J. A. Hupp, "Measured Performance of an Ethernet Local Network", Xerox Technical Report, February 1980.

[SHOC82] J. F. Shoch, Y. K. Dalal, D. A. Redell, "Evolution of the Ethernet Local Computer Network", Computer, August 1982, pages 10-27.

[SMIT82] M. F. Smith, R. J. Loader, "Experimenting with Local Networks using an Integrated Circuit Data Link Controller", Microprocessors and Microsystems, Vol. 6, No. 1, January-February 1982, pages 9-13.

[THUR82] K. J. Thurber, ed., "The LOCALNetter Designer's Handbook", 1982 Edition, Architecture Technology Corporation.

[YAUS82] S. S. Yau, S. M. Shatz, "On Communication in the Design of Software Components of Distributed Computer Systems", Proceedings of

the 3rd International Conference on Distributed Computing Systems, 1982, pages 280-287.

[YOSH82] S. Yoshitake, M. Mashio, S. Ideguchi, M. Katsumata, "Method for Testing Data Communication Products That Implement Standard Protocols", Proceedings of the 3rd International Conference on Distributed Computing, pages 742-747.

APPENDIX A

SCI-NET SOFTWARE LISTING

```
          OPT PAG
          TTL     ACIA NETWORK PROGRAM - VER. 92
          PAG 1

* SCI-net driver

* Author..............  Rene S. Hollan
* Date Written........  July 1983
* Last modification...  December 15, 1983


*         ACIA EQUATES

TDR       EQU     $E018     TRANSMIT DATA REGISTER
RDR       EQU     $E018     RECEIVER DATA REGISTER
STATUS    EQU     $E019     ACIA STATUS REGISTER
CMDR      EQU     $E01A     ACIA COMMAND REGISTER
CTRR      EQU     $E01B     ACIA CONTROL REGISTER
RDRF      EQU     $08       RECEIVE DATA REGISTER FULL
TDRE      EQU     $10       TRANSMIT DATA REGISTER EMPTY
ICTREX    EQU     $10       INIT ACIA TO EXTERNAL CLOCK
ICTR19    EQU     $1F       INIT ACIA 19200 BAUD
ICTR96    EQU     $1E       INIT ACIA 9600 BAUD
ICTR48    EQU     $1C       INIT ACIA 4800 BAUD
ICTR03    EQU     $16       INIT ACIA 300 BAUD
ENABLE    EQU     $09       ENABLE IRQ WHEN ACIA RDRF
DIS       EQU     $0B       DISABLE IRQ INTERRUPTS
ERROR     EQU     $07       ERROR BITS IN STATUS REGISTER
PARITY    EQU     $7F       ASCII DATA BITS
PARERR    EQU     $01       PARITY ERROR BIT
FRAERR    EQU     $02       FRAMING ERROR BIT
OVEERR    EQU     $04       OVERRUN ERROR BIT

*         TIMER EQUATES

TCR1      EQU     $E038     TIMER CONTROL REGISTER 1
TCR2      EQU     $E039     TIMER CONTROL REGISTER 2
TCR3      EQU     $E038     TIMER CONTROL REGISTER 3
TSTAT     EQU     $E039     TIMER STATUS REGISTER
TCTR1     EQU     $E03A     TIMER COUNTER 1 (NOT USED)
```

```
TCTR2   EQU     $E058       TIMER COUNTER 2
TCTR3   EQU     $E05A       TIMER COUNTER 3

*       STATUS FLAGS

S.XMIT  EQU     $01         TRANSMITTER ON LINE
S.LXMT  EQU     $02         LOCAL XMIT IN PROGRESS
S.WACK  EQU     $04         AN ACK CAME IN
S.SACK  EQU     $08         SENDING AN ACK
S.DDLE  EQU     $10         DON'T DOUBLE DLE ON SEND

*       USER STATUS FLAGS

S.MREC  EQU     $01         MESSAGE RECIEVED AND IN BUFFER
S.OERR  EQU     $02         MESSAGE OVERRUN ERROR
S.FCMD  EQU     $04         FLEX COMMAND
S.CMDC  EQU     $08         FLEX COMMAND CHARACTERS COME FROM US

*       SYSTEM EQUATES

GETHEX  EQU     $CD42
INCHE   EQU     $F806
INCH    EQU     $CD09
LINBUF  EQU     $C080
LINPTR  EQU     $CC14
OUTCH   EQU     $F80A
PCRLF   EQU     $F80E
PSTRNG  EQU     $F810
PDATA   EQU     $F80C
STAT    EQU     $CD4E
WARMS   EQU     $CD03

*       SYSTEM PATCH EQUATES

FINCH   EQU     $CF14       FLEX COMMAND CHARACTER INPUT ROUTINE
FINTR   EQU     $D3CB       FLEX INTERRUPT SERVICE ROUTINE

*       ASCII EQUATES

NUL     EQU     $00
SOH     EQU     $01
STX     EQU     $02
ETX     EQU     $03
EOT     EQU     $04
ACK     EQU     $06
LF      EQU     $0A
CR      EQU     $0D
DLE     EQU     $10
SYN     EQU     $16
CLO     EQU     $4C
OPE     EQU     $4F
CMD     EQU     $46

*       JUMP VECTORS
```

```
        ORG     $2000

        JMP     READ
        JMP     WRITE
        JMP     OPEN
        JMP     CLOSE
        JMP     READW
        JMP     SPMESS
        JMP     OPENW

*       DATA    AREA

UFLAGS  FCC     $00     USER STATUS FLAGS
ID      RMB     1       STATION ID #
PRIO    RMB     1       STATION PRIORITY
SRC     RMB     1       SOURCE OF INCOMING MESSAGE
MSRC    RMB     1       SOURCE OF MESSAGE IN BUFFER
DEST    RMB     1       DESTINATION OF OUTGOING MESSAGE
MSGC    RMB     1       MESSAGE COUNT
CHAN    FCB     0       CHANNEL CURRENTLY OPEN
DLEC    RMB     1       LAST DLE'D CHARACTER
RMSG    RMB     1       LAST REC'D MESSAGE NUMBER
RMSG1   RMB     1       CURRENT REC'D MESSAGE NUMBER
FLAGS   FCB     $00     STATUS FLAGS
TEMP    RMB     1
FUBAR1  FCB     1
BUFFER  RMB     256     MESSAGE BUFFER
BUFLEN  RMB     1       MESSAGE BUFFER LENGTH
RLEN    RMB     1       RECEIVED MESSAGE BUFFER LENGTH
LEN     RMB     1       INCOMMING MESSAGE LENGTH
TRANS   FDB     0
MSENT   FDB     0
MRECV   FDB     0
NINC    FCB     0       CHARACTERS IN COMMAND PUSH BUFFER
NINBP   FDB     NINB    POINTER TO NEXT CHARACTER TO GET
NINB    EQU     BUFFER  COMMAND PUSH BUFFER
FUB1 RMB 1
FUB2 RMB 1
FUB3 RMB 1

*       TEST PROGRAM BEGINS HERE


*       ACIA INITIALIZATION

START   JSR     GETHEX  GET THE ID OF THIS STATION
        TFR     X,D
        STB     ID      AND STORE IT
        STA     STATUS  RESET ACIA
        LDA     #ICTREX
        STA     CTRR    PROGRAM ACIA FOR 112,000 BAUD
        LDA     #ENABLE AND IRQ INTERRUPTS ENABLED
```

```
        STA     CMDR

*       TIMER INITIALIZATION

        LDA     #$60    COUNTER2: SINGLE SHOT, IRQ
        STA     TCR2
        LDA     #$81    COUNTER3: CONTINUOUS, OUTPUT, PRE-SCALED
        STA     TCR3
        LDA     #$61    SET UP COUNTER 1
        STA     TCR2
        LDA     #$01    RESET AND DISABLE COUNTERS
        STA     TCR1
        LDX     #$0A    BYTE TIME TO COUNTER 3
        STX     TCTR3
        JMP     WARMS   RETURN TO FLEX

*       CHARACTER MESSAGES

MSG2    FCC     'A MESSAGE FOR US WAS LOST',$D,$A,$4

FINCH1  EQU     *

*       SET IRQ JUMP

        ORG     FINTR
        JMP     INTR

*       SET FINCH PATCH

        ORG     FINCH
        JMP     NINCH

        ORG     FINCH1
NINCH   LDA     NINC    GET CHARACTERS IN PUSH BUFFER
        BEQ     FINCH2  NONE LEFT

        LDA     UFLAGS  GET USER FLAG BITS
        CMPX    #LINBUF ARE WE AT THE BASE OF THE USER LINE BUFFER?
        BNE     FINCH3  NOPE

        ORA     #S.CMDC AT BASE OF BUFFER SO SET BIT SAYING CMD.
        STA     UFLAGS  CHARACTERS COME FROM US.

FINCH3  ANDA    #S.CMDC DO COMMAND CHARACTERS COME FROM US?
        BEQ     FINCH2  NOPE

        LDA     [NINBP] GET CHARACTER
        PSHS    A,B     SAVE THESE FOR A WHILE
        LDD     NINBP   ADVANCE BUFFER POINTER
        ADDD    #$0001
        STD     NINBP   AND REPLACE IT
        DEC     NINC    DECREMENT CHARACTER COUNT
        BNE     NINCH1  STILL SOME MORE TO PROCESS
```

```
        LDD     #NINB    RESET BUFFER POINTER
        STD     NINBP

        ORCC    #$10     DISABLE INTERRUPTS
        LDA     UFLAGS   GET USER FLAGS
        ANDA    #$FF-S_MREC-S_OERR-S_FCMD-S_CMDC
        STA     UFLAGS   UPDATE USER FLAGS
        ANDCC   #$EF     RESTORE INTERRUPTS

NINCH1  PULS    A,B      RESTORE REGISTERS
        RTS

FINCH2  JSR     STAT     GET TERMINAL STATUS
        BEQ     NINCH    NO CHARACTER YET, CHECK LINK BUFFER
        JSR     INCH     GET CHARACTER FROM CONSOLE
        JMP     FINCH+3  AND RETURN TO FLEX'S INPUT COMMAND CHAR RTN


*       INTERRUPT PROCESSING ROUTINE

INTR    LDA     TSTAT    GET TIMER STATUS
        LBMI    TIMSER   INTERRUPT SERVICE ROUTINE
        LDA     STATUS   GET ACIA STATUS
        ANDA    #RDRF    IS IT ACIA?
        BNE     READ_I   YUP!
        LDA     RDR
        RTI

*       READ ROUTINE

READ_I  CLR     DLEC     NO DLE'D CHARACTER
        LBSR    RDBYTE   GET BYTE FROM ACIA
        CMPA    #SYN     IS IT A SYN?
        BEQ     READ1    YUP
        RTI

READ1   LBSR    RDBYTE   GET BYTE FROM ACIA
        CMPA    #SYN     IS IT ANOTHER SYN?
        BEQ     READ1    YUP, KEEP WAITING

        CMPA    #DLE     IS IT A DLE?
        BEQ     READ2    YUP!
        RTI

READ2   LBSR    RDBYTE   GET BYTE FROM ACIA
        CMPA    #SOH     IS IT A SOH?
        BEQ     READ3    YUP!
        RTI

READ3   LBSR    RDBYTE   GET THE SOURCE FROM ACIA
        STA     SRC      SAVE SOURCE ADDRESS
        LBSR    RDBYTE   GET MESSAGE NUMBER BEING SENT
        STA     RMSG1    AND SAVE IT
        LBSR    RDBYTE   GET ADDRESS FROM ACIA
```

```
        CMPA    ID          IS IT FOR ME?
        BNE     READ31      NOPE, SET UP TIMER TO IGNORE MESSAGE

        TST     CHAN        ARE WE TALKING TO ONLY ONE STATION?
        BEQ     READ4       NOPE, IT'S A FREE FOR ALL
        LDA     SRC         AM I LISTENING TO THIS GUY?
        CMPA    CHAN
        BEQ     READ4       YUP!, (LUCKY FOR HIM)

*       IT AIN'T FOR ME SO WE DISABLE ACIA INTERRUPTS FOR THE RIGHT
* LENGTH OF TIME

READ31  LDA     FLAGS       CHECK IF WE'RE WAITING FOR AN ACK
        ANDA    #S.WACK
        BNE     READ76      YUP, DON'T SET UP TIMER
        LBSR    RDBYTE      GET THE LENGTH OF THE MESSAGE
        LDB     #DIS        DISABLE INTERRUPTS FROM ACIA
        STB     CMDR
        CLR     TCTR2       CLEAR MSB OF TIMER BYTE COUNTER
        STA     TCTR2+1     STORE COUNT INTO LSB OF TIMER BYTE COUNTER
        CLR     TCR1        AND AWAY WE GO!

        LDB     FLAGS       SET XMIT IN PROGRESS FLAG
        ORB     #S.XMIT
        STB     FLAGS
        RTI

READ4   LDA     UFLAGS      GET USER STATUS FLAGS
        ANDA    #S.MREC     IS THERE A MESSAGE IN OUR RECEIVE BUFFER?
        BEQ     READ41      NOPE, EVERYTHING'S COOL

        LDA     UFLAGS      GET USER FLAG BITS
        ORA     #S.OERR     SET OVERRUN ERROR BIT
        STA     UFLAGS      REPLACE USER FLAG BITS
        BRA     READ31      AND IGNORE MESSAGE

READ41  LBSR    RDBYTE      GET LENGTH OF MESSAGE
        STA     LEN         AND SAVE IT
        LBSR    RDBYTE      GET *DLE* (HOPEFULLY)
        CMPA    #DLE        IS IT DLE?
        BEQ     READ5       YUP!, SO FAR, SO GOOD

        LDX     #MSG2       TELL HIM WE LOST IT
        JSR     [PSTRNG]
        RTI

READ5   LBSR    RDBYTE      ,GET *STX* (HOPEFULLY)
        CMPA    #STX        IS IT STX?
        BEQ     READ6       YUP! (MAN!, WE'RE COOKING NOW!)
        RTI                 OH NO!, WE'RE IN TROUBLE

READ6   LDX     #BUFFER     X --> BUFFER
        CLR     BUFLEN      CLEAR BUFFER LENGTH
```

```
READ62  LBSR    RDBYTE      GET A CHARACTER
        CMPA    #DLE        IS IT DLE?
        BEQ     READ7       YUP, WE MUST HANDLE IT

READ65  STA     0,X+        SAVE THE CHARACTER IN A BUFFER
        INC     BUFLEN      INCREMENT BUFFER LENGTH
        BRA     READ62      KEEP GOING

READ7   LBSR    RDBYTE      GET THE DLE'D CHARACTER
        CMPA    #ETX        IS IT THE END OF OUR MESSAGE?
        BEQ     READ75

        STA     DLEC        SAVE LAST DLE'D CHARACTER
        BRA     READ65      AND LOOP

READ75  LDA     DLEC        GET LAST DLE'D CHARACTER
        CMPA    #ACK        WAS IT AN ACK?
        BNE     READ77      NOPE, KEEP CHECKING

        LDA     SRC         GET THE SOURCE OF THE ACK
        CMPA    DEST        IS IT FROM THE RIGHT GUY?
        BNE     READ76      NOPE, IT'S NOT

        LDA     FLAGS       CLEAR WAIT FOR ACK BIT IN FLAGS
        ANDA    #$FF-S.WACK
        STA     FLAGS
READ76  RTI

READ77  LDA     FLAGS     GET STATUS FLAGS
        ANDA    #S.WACK   ARE WE WAITING FOR AN ACK?
        BNE     READ76    YUP, MUST SETTLE THIS FIRST

        LDA     DLEC      RESTORE DLE'D CHARACTER
        CMPA    #OPE      IS IT AN OPEN REQUEST?
        BNE     READ78    NOPE

        TST     CHAN      CAN WE OPEN A CHANNEL?
        BEQ     REA771    YUP!
        LDA     SRC       GET THE SOURCE OF THE REQUEST
        CMPA    CHAN      DO WE ALREADY HAVE A CHANNEL OPEN TO HIM?
        BEQ     READ85    YUP!, JUST SEND AN ACK BACK
        RTI               HE'S GOTTA WAIT

REA771  LDA     SRC       SOURCE TO OPEN CHANNEL TO
        STA     CHAN      OPEN A CHANNEL TO HIM
        CLR     RMSG      CLEAR RECEIVED MESSAGE COUNTER
        BRA     READ85    AND SEND AN ACK BACK

READ78  CMPA    #CLO      IS THIS A REQUEST TO CLOSE?
        BNE     REA781    NOPE, TREAT IT AS A NORMAL MESSAGE

        LDA     SRC       GET THE SOURCE OF THE REQUEST
        CMPA    CHAN      IS IS THE SAME AS THE CURRENT OPEN CHANNEL?
        BNE     READ76    NOPE, BUZZ OFF BUSTER!
```

```
        CLR     CHAN      CLEAR THE CHANNEL
        BRA     READ85    AND SEND AN ACK BACK

REA781  CMPA    #CMD      IS IT A FLEX COMMAND?
        BNE     READ80    NOPE, TREAT AS NORMAL MESSAGE

        LDA     RMSG1     GET CURRENT MESSAGE NUMBER
        CMPA    RMSG      IS IT THE SAME AS THE LAST ONE?
        BEQ     READ85    YUP, HE LOST OUT ACK

        STA     RMSG      UPDATE MESSAGE NUMBER

        LDA     LEN       GET LENGTH OF MESSAGE
        SUBA    #$06      SUBTRACT LENGTH OF DLE CMD
        STA     NINC      STORE IN COMMAND PUSH BUFFER LENGTH
        LDA     UFLAGS    GET USER FLAGS
        ORA     #S.MREC+S.FCMD  SET MESSAGE RECEIVED BIT
        STA     UFLAGS    AND UPDATE USER FLAGS

        BRA     READ85    SEND AN ACK BACK

READ80  LDA     RMSG1     GET MESSAGE NUMBER JUST RECEIVED
        CMPA    RMSG      IS IT THE SAME AS THE LAST ONE?
        BEQ     READ85    YUP, HE LOST OUR ACK

        STA     RMSG      UPDATE MESSAGE NUMBER

        LDA     UFLAGS    GET USER STATUS FLAGS
        ORA     #S.MREC   SET RECEIVED MESSAGE BIT
        STA     UFLAGS
        LDA     SRC       GET SOURCE OF MESSAGE.
        STA     MSRC      AND SAVE IT
*       LDA     #EOT      TERMINATE MESSAGE IN BUFFER
*       STA     0,X
*       LDX     #BUFFER   X --> MESSAGE
*       JSR     [PDATA]

READ85, LDA     FLAGS     GET FLAG BITS
        ORA     #S.SACK+S.DDLE  WE'RE GOING TO SEND AN ACK
        STA     FLAGS
        LDA     #DIS      DISABLE ACIA INTERRUPTS
        STA     CMDR
        LDA     STATUS    CLEAR ANY PENDING INTERRUPTS
        LDA     RDR       AND CLEAR RDRF FLAG
        LDA     #2        TWO BYTES TO SEND
        LDX     #READ81   X --> DLE ACK
        LDB     SRC       B' = SOURCE OF MESSAGE TO ACK
*       ANDCC   #$EF      TURN INTERRUPTS ON!!!!!!
        LBSR    WRITE     SEND THE ACK
*       ORCC    #$10      TURN INTERRUPTS OFF (WHEW!)
        LDA     #ENABLE   RESTART ACIA ARQ'S
        STA     CMDR

        PSHS    X
```

```
        LDX     MRECV
        LEAX    1,X
        STX     MRECV
        PULS    X
        RTI

READ81  FCC     DLE,ACK

*       RDBYTE: GET A CHARACTER FROM ACIA AND RETURN IT IN A

RDBYTE  CLR     FUBAR1
RDBYT2  LDA     STATUS      WAIT FOR A CHARACTER
        ANDA    #RDRF       IS THERE ONE?
        BNE     RDBYT1      YUP!

        INC     FUBAR1
        BNE     RDBYT2

        LDX     #RDBYT3
        JSR     [PSTRNG]
        LDA     RDR
        PULS    X           POP A DUMMY 16 BIT VALUE
        RTI

RDBYT1  LDA     RDR         GET THE CHARACTER
        RTS

RDBYT3  FCC     'TIMEOUT IN RDBYTE!',$04

*       TIMSER: TIMER INTERRUPT SERVICE ROUTINE

TIMSER  LDA     #$01        STOP THE TIMER
        STA     TCR1
        LDA     FLAGS       CHECK IF A LOCAL TRANSMIT IS IN PROGRESS
        ANDA    #S.LXMT
        BNE     TIMS01      YUP, DON'T ENABLE ACIA IRQ'S

        LDA     #ENABLE     REENABLE ACIA INTERRUPTS
        STA     CMDR

TIMS01  LDA     FLAGS       CLEAR XMIT IN PROGRESS FLAG
        ANDA    #$FF-S.XMIT
        STA     FLAGS
        RTI


*       READ - TRANSFER CURRENT MESSAGE IN READ BUFFER INTO USER
*              BUFFER.

*       ENTRY: A = LENGTH OF USER BUFFER
*              X = ADDRESS OF USER BUFFER
```

```
*       EXIT:  A = LENGTH OF MESSAGE IN USER BUFFER
*              B = SOURCE OF MESSAGE
*              X = ADDRESS OF USER BUFFER

READ    ORCC   #$10       TURN OFF INTERRUPTS
        PSHS   X,Y        SAVE REGISTERS
        STA    RLEN       SAVE MAX. LENGTH OF MESSAGE
        LDB    BUFLEN     B = BUFFER LENGTH
        LDY    #BUFFER    Y --> MESSAGE BUFFER

READ.0  TSTA              HAVE WE EXCEEDED MAX LENGTH?
        BEQ    READ.1     YUP
        TSTB              HAVE WE EXCEEDED MESSAGE LENGTH?
        BEQ    READ.2     YUP

        PSHS   B          SAVE THIS
        LDB    0,Y+       GET A BYTE FROM THE MESSAGE BUFFER
        STB    0,X+       AND STORE IT IN THE USER BUFFER
        PULS   B          RESTORE THIS AGAIN

        DECA              DECREMENT MAX COUNTER
        DECB              DECREMENT ACTUAL COUNTER
        BRA    READ.0     AND GO MOVE THE NEXT BYTE

READ.2  LDA    BUFLEN     GET MESSAGE LENGTH
        BRA    READ.3

READ.1  LDA    RLEN       GET MAX. MESSAGE LENGTH

READ.3  LDB    UFLAGS     TURN OFF MESSAGE IN BUFFER BIT
        ANDB   #$FF-S.MREC-S.OERR
        STB    UFLAGS

        LDB    MSRC       GET SOURCE OF MESSAGE
        PULS   X,Y        RESTORE REGISTERS
        ANDCC  #$EF       TURN ON INTERRUPTS
        RTS


*       READW - WAITS TILL A MESSAGE IS AVAILABLE, THEN CALLS
*               READ.


READW   PSHS   A          SAVE THIS REGISTER
READW1  LDA    UFLAGS     GET USER STATUS FLAGS
        ANDA   #S.MREC    DO WE HAVE A MESSAGE?
        BEQ    READW1     NOT YET

        PULS   A          RESTORE THIS REGISTER
        BSR    READ       GO CALL READ
        RTS
```

```
*       WRITE ROUTINE

*       ON ENTRY X --> BUFFER TO SEND, A = LENGTH, B = DEST.

WRITE   INC     MSGC        INCREMENT OUTGOING MESSAGE COUNT
        STB     DEST        SAVE DESTINATION ADDRESS

WRIT16  PSHS    A,B,X       SAVE REGGIES
        PSHS    B           SAVE DESTINATION
        ORCC    #$10        TURN OFF ALL INTERRUPTS
        LDB     FLAGS       MARK TRANSMISSION IN PROGRESS
        ORB     #S.LXMT
        STB     FLAGS
        LDB     #DIS        DISABLE ACIA IRQ'S
        STB     CMDR
*       ANDCC   #$EF        TURN INTERRUPTS BACK ON

        PSHS    X
        LDX     MSENT
        LEAX    1,X
        STX     MSENT
        PULS    X

        LDB     FLAGS       IS THERE A TRANSMISSION ALREADY IN
PROGRESS?
        ANDB    #S.XMIT
        BEQ     WRITE5      NOPE!, WE DON'T HAVE TO WAIT FOR IT TO
FINISH

WRIT51  LDB     TSTAT       GET TIMER STATUS
        ANDB    #$02        HAS THE TIMER TIMED OUT?
        BEQ     WRIT51      NOT YET (HO HUM)

        LDA     #$01
        STA     TCR1        STOP THE TIMER

*WRITE5 LBSR    DELAY       MAKE SURE THAT THE LINK IS FREE
WRITE5  LBSR    LENGTH      GET TRUE LENGTH IN A
        LBSR    DELAY       MAKE SURE THAT THE LINK IS FREE
        PULS    B           RESTORE DESTINATION
        PSHS    A          AND SAVE LENGTH

WRITE4  PSHS    X
        LDX     TRANS
        LEAX    1,X
        STX     TRANS
        PULS    X

        LDA     #SYN        SEND THE START OF THE MESSAGE
        LBSR    WRBYTE      WRITE THE BYTE
        LBSR    WRBYTE      AND AGAIN
        LBSR    WRBYTE
        LBSR    WRBYTE
        LDA     #DLE        START OF HEADER
```

```
        LBSR    WRBYTE      DLE
        LDA     #SOH
        LBSR    WRBYTE      SOH

WRIT41  LDA     STATUS      WAIT ON TDRE
        STA     FUB1
        ANDA    #TDRE
        BEQ     WRIT41
        CLR     FUB1
        LDA     RDR         CLEAR ANY OLD RDRF
        LBSR    WRDRF       WAIT ON RDRF OR TIMEOUT
        BEQ     WRITDL      TIMEOUT, SO DELAY
        LDA     RDR         CLEAR LAST BYTE SENT

        LDA     ID          SEND TRANSMITTER ID
        LBSR    WRBYTE
        LBSR    WRDRF       WAIT ON RDRF OR TIMEOUT
        BEQ     WRITDL      TIMEOUT, SO DELAY
        LDA     RDR         GET BYTE THAT WE SENT
        CMPA    ID          DID WE HAVE A COLLISION (ERE?
        BEQ     WRITE3      NOPE


WRITDL  LBSR    DELAY       WE HAD A COLLISION, DELAY A RANDOM TIME
        LDA     #07
        JSR     $CD0F
        BRA     WRITE4      AND TRY AGAIN

WRITE3  LDA     MSGC        SEND MESSAGE COUNT
        LBSR    WRBYTE
        TFR     B,A         SEND ADDRESS
        LBSR    WRBYTE
        PULS    A           GET LENGTH
        TFR     A,B         SAVE IT
        SUBB    #4          OFFSET IT
        LBSR    WRBYTE      SEND IT
        LDA     #DLE        DLE
        LBSR    WRBYTE
        LDA     #STX        STX
        LBSR    WRBYTE

WRITE1  TSTB                ARE WE AT THE END?
        BEQ     WRITE2      YUP

        LDA     0,X+        GET A BYTE
        LBSR    WRBYTE      SEND IT
        DECB                DECREMENT COUNT
        PSHS    A           SAVE THE BYTE JUST SENT
        LDA     FLAGS       GET FLAGS
        ANDA    #$DDLE      ARE WE NOT DOUBLING DLE ON SEND?
        BEQ     WRIT15      NOPE
        PULS    A           POP OF THE STACK
        BRA     WRITE1
```

```
WRIT15 PULS   A          POP BYTE JUST SENT
       CMPA   #DLE       WAS IT A DLE?
       BNE    WRITE1     NOPE, ALL O.K.

       LBSR   WRBYTE     SEND IT AGAIN
       BRA    WRITE1     AND LOOP FOR MORE

WRITE2 LDA    #DLE       TERMINATE MESSAGE
       LBSR   WRBYTE     DLE
       LDA    #ETX       END OF TEXT
       LBSR   WRBYTE     ETX
       LDA    FLAGS      MARK TRANSMISSION COMPLETE
       ANDA   #$FF-S.LXMT
       STA    FLAGS
       LDA    STATUS
       LDA    RDR        CLEAR RDRF FLAG
       LDA    FLAGS      CHECK IF WE SEND AN ACK BACK
       ANDA   #S.SACK
       BEQ    WRITE7         NOPE, WE MUST WAIT FOR AN ACK TO COME
IN

       LDA    FLAGS          CLEAR S.SACK BIT, S.DDLE BIT
       ANDA   #$FF-S.SACK-S.DDLE
       STA    FLAGS
       PULS   A,B,X
       RTS

WRITE7 LDD    #$100      SET UP TIMER FOR INTERRUPT
       STD    TCTR2
       LDA    FLAGS      SET TIMER COUNTING BIT
       ORA    #S.WACK+S.XMIT
       STA    FLAGS
       CLR    TCR1       START THE TIMER GOING

       LDA    #ENABLE    ENABLE ACIA INTERRUPTS
       STA    CMDR
       LDA    STATUS
       LDA    RDR
       ANDCC  #$EF       START ALL INTERRUPTS

WRITE6 LDA    FLAGS     WAIT FOR AN ACK TO COME BACK
       TFR    A,B       SAVE THIS FOR ANOTHER CHECK
       ANDB   #S.XMIT       DID THE TIMER COUNT DOWN YET?
       BEQ    WRITE8    YUP!

       ANDA   #S.WACK
       BNE    WRITE6    STILL WAITING FOR AN ACK

       LDA    #$01       STOP TIMER
       STA    TCR1
       ORCC   #$10       DISABLE IRQS
       LDA    FLAGS      TURN OFF TIMER BIT
       ANDA   #$FF-S.DDLE-S.XMIT
       STA    FLAGS
```

```
        LDA     STATUS
        LDA     RDR
        ANDCC   #$EF        ENABLE IRQS
        PULS    A,B,X
        RTS

WRITE8  LDA     #$01        CLEAR TIMER STATUS
        STA     TCR1

        ORCC    #$10        TURN INTERRUPTS OFF
        LDA     FLAGS
        ANDA    #$FF-S.XMIT
        STA     FLAGS
        LDA     STATUS
        LDA     RDR
*       ANDCC   #$EF        TURN INTERRUPTS BACK ON
        LDA     #$41
        JSR     $CD0F
        PULS    A,B,X       RESTORE REGGIES
        LBRA    WRIT16      AND TRY AGAIN

*       WRDRF - WAIT FOR RDRF OR TIMEOUT, A IS CLEARED IF TIMEOUT.

WRDRF   PSHS    B           SAVE B
        CLRB                AND CLEAR OUT COUNTER
WRDRFL  LDA     STATUS      GET ACIA STATUS
        ANDA    #RDRF
        BNE     WRDRFX      WE GOT AN RDRF!

        INCB                INCREMENT COUNTER
        BNE     WRDRFL      LOOP SOME MORE

        PULS    B           RESTORE B
        CLRA                CLEAR RESULT
        RTS

WRDRFX  PULS    B           RESTORE B
        TFR     A,A         SET CONDITION CODES
        RTS                 AND RETURN


*       WRBYTE - WRITE A BYTE IN A AND RETURN READ BYTE IN A

WRBYTE  PSHS    B           SAVE B
WRB01   LDB     STATUS      GET THE ACIA STATUS
        ANDB    #TDRE       CAN WE SEND?
        BEQ     WRB01       NOT YET

        STA     TDR         SEND THE BYTE
        PULS    B           RESTORE B
        RTS
```

```
WRB03   LDA     RDR         READ THE BYTE
        LDB     #$01
        STB     TCR1
*       LDB     FLAGS
*       ANDB    #$FF-S_XMIT
*       STB     FLAGS
        PULS    B           RESTORE B
        RTS


*       DELAY - DELAY FOR A TIME

DELAY   PSHS    A           SAVE THIS

DELAY1  LDA     RDR         CLEAR RDRF FLAG
        LDA     PRIO        A = PRIO OF THIS STATION
        ANDA    #$0F
        ORA     #$08
        INC     PRIO

        PSHS    A           SAVE DELAY TIME
        LDA     FLAGS       GET FLAGS
        ANDA    #S_SACK     ARE WE SENDING AN ACK?
        BEQ     DELAY3      NOPE

        PULS    A           RESTORE DELAY TIME
        ASRA                AND DIVIDE BY TWO
        BRA     DELAY4

DELAY3  PULS    A           RESTORE DELAY TIME

DELAY4
        CLR     TCTR2       USE THIS AS A DELAY COUNT
        STA     TCTR2+1
*       LDA     FLAGS       SET XMIT IN PROGRESS FLAG
*       ORA     #S_XMIT
*       STA     FLAGS
        CLR     TCR1        AND START THE TIMER

DELAY2  LDA     TSTAT       CHECK IF TIMER TIMED OUT
        ANDA    #$02
        BEQ     DELAY2      NOT YET

        LDA     STATUS      WAS THE LINK BUSY?
        ANDA    #RDRF
        BNE     DELAY1      YUP!

        LDA     #$01
        STA     TCR1
*       LDA     FLAGS
*       ANDA    #$FF-S_XMIT
*       STA     FLAGS
        PULS    A           RESTORE THIS
```

```
        RTS

*       LENGTH - RETURN TRUE LENGTH OF MESSAGE AT X, LENGTH IN A, IN
*       REGISTER A

LENGTH  PSHS    X           SAVE X
        TFR     A,B         B = CURRENT LENGTH
        ADDB    #$04        ADD OFFSET
        STB     TEMP        STORE IT

LEN01   TSTA                ARE WE AT THE END OF THE MESSAGE?
        BEQ     LEN02       YUP

        LDB     0,X+        GET A BYTE
        DECA                DECREMENT LENGTH
        CMPB    #DLE        IS IT A DLE?
        BNE     LEN01       NOPE
        LDB     FLAGS       ARE WE DOUBLING DLE ON SEND?
        ANDB    #SNDDLE
        BNE     LEN01

        INC     TEMP        YUP, ADJUST LENGTH
        BRA     LEN01

LEN02   LDA     TEMP        GET TRUE LENGTH IN A
        PULS    X           RESTORE X
        RTS                 AND GO HOME

*       OPEN - OPEN A CHANNEL TO THE STATION WHOSE ID IS IN A.

OPEN    ORCC    #$10        TURN OFF INTERRUPTS
        TST     CHAN        HAVE WE CURRENTLY GOT AN OPEN CHANNEL TO
        BEQ     OPEN1       ANYONE? NOPE?, ALL'S O.K., THEN
        CMPA    CHAN        HAVE WE GOT A CHANNEL TO THE RIGHT GUY?
        BEQ     OPEN2       YUP!, NOTHING TO DO

        CLRA                YA CAN'T DO THAT! GOTTA CLOSE FIRST!

OPEN2   ANDCC   #$EF        TURN INTERRUPTS BACK ON
        RTS

OPEN1   STA     CHAN        WE WANT TO OPEN A CHANNEL TO THIS GUY
        CLR     MSGC        INITIALIZE THIS
        COM     MSGC        TO -1

        PSHS    A,B,X       SAVE HIS PRECIOUS REGGIES
        LDX     #OPEN3      OPEN MESAGE
        TFR     A,B         B = DESTINATION
        LDA     #$02        MESSAGE LENGTH
        BSR     SPMESS      SEND SPECIAL MESSAGE
        PULS    A,B,X       RESTORE REGGIES
        RTS
```

```
OPEN3   FCC     DLE,OPE

*       OPENW - OPEN A CHANNEL BUT WAIT FOR LINK TO BECOME FREE.

OPENW   PSHS    A           SAVE CHANNEL NUMBER
        BSR     OPEN        TRY TO OPEN A CHANNEL
        TSTA                GET STATUS
        BNE     OPENW1      IT WORKED!

        PULS    A           RESTORE CHANNEL NUMBER
        BRA     OPENW       AND TRY AGAIN

OPENW1  PULS    A           ALL DONE
        RTS


*       CLOSE - CLOSE THE CHANNEL CURRENTLY OPEN

CLOSE   ORCC    #$10        TURN OFF INTERRUPTS
        TST     CHAN        HAVE WE GOT A CHANNEL OPEN TO SOMEONE?
        BNE     CLOSE1      YUP, GOTTA CLOSE IT.

        CLRA                TELL HIM NO CHANNEL OPEN
        ANDCC   #$EF        TURN INTERRUPTS BACK ON
        RTS

CLOSE1  PSHS    A,B,X       SAVE HIS PRECIOUS REGGIES (AS IF HE CAN'T
        LDX     #CLOSE2     SAVE THEM HIMSELF!)
        LDB     CHAN        B = GUY TO SEND MESSAGE TO
        CLR     CHAN        CLEAR CHANNEL
        LDA     #$02        MESSAGE LENGTH
        BSR     SPMESS      SEND THE SPECIAL MESSAGE
        PULS    A,B,X       RESTORE REGGIES
        RTS

CLOSE2  FCC     DLE,CLO


*       SPMESS - SEND A SPECIAL MESSAGE.

*                A = MESSAGE LENGTH
*                B = DESTINATION, X = ADDRESS OF MESSAGE

*                A SPECIAL MESSAGE IS SENT WITH DLE'S NOT DOUBLED.
*                THIS ROUTINE MUST BE ENTERED WITH INTERRUPTS OFF.
*                IT WILL EXIT WITH INTERRUPTS ON UNLESS S.ACK WAS
*                SET IN THE STATUS BYTE.

SPMESS  PSHS    A           SAVE MESSAGE LENGTH
        LDA     FLAGS       GET STATUS BITS
        ORA     #S.DDLE
        STA     FLAGS
        PULS    A           RESTORE MESSAGE LENGTH
```

```
LBSR   WRITE     SEND THE MESSAGE
RTS

END    START
```

APPENDIX B

GPSS SIMULATION PROGRAM

```
 SIMULATE 4.0
*
* GPSS SIMULATION OF SCI-NET
*
* AUTHOR..... RENE S. HOLLAN
* DATE WRITTEN... MARCH 21, 1984
*
*    SIMULATION OF FOUR TRANSMITTERS TO FOUR RECEIVERS WITH
* WAIT FOR ACK TIMES OF 2500, 5000, 10000, 20000 BIT TIMES.
*
*
EXP FUNCTION RN1,C24

0,0/.1,.104/.2,.222/.3,.355/.4,.509/.5,.69/.6,.915/.7,1.2/.75,1.38/

.8,1.6/.84,1.83/.88,2.12/.9,2.3/.92,2.52/.94,2.81/.95,2.99/.96,3.2/
.97,3.5/.98,3.9/.99,4.6/.995,5.3/.998,6.2/.999,7/.9997,8
*
DLY FUNCTION RN1,C8                    DELAY FUNCTION
.125,80/.25,90/.375,100/.5,110/
.625,120/.75,130/.875,140/1,150
*
LEN FUNCTION PB2,E2                    MESSAGE
LENGTH (REGULAR/ACK)
.5,XH$LEN/1,60
*
MAP FUNCTION PB1,S32,L                 MAP STATION ID TO CPU
FACILITY
1,STA1/2,STA2/3,STA3/4,STA4/
5,STA5/6,STA6/7,STA7/8,STA8/
9,STA9/10,STA10/11,STA11/12,STA12/
13,STA13/14,STA14/15,STA15/16,STA16/
17,STA17/18,STA18/19,STA19/20,STA20/
21,STA21/22,STA22/23,STA23/24,STA24/
25,STA25/26,STA26/27,STA27/28,STA28/
29,STA29/30,STA30/31,STA31/32,STA32/
*
WCN FUNCTION PB1,S32,C                 MAP STATION ID TO ACK
CHAIN
1,WCK1/2,WCK2/3,WCK3/4,WCK4/
5,WCK5/6,WCK6/7,WCK7/8,WCK8/
9,WCK9/10,WCK10/11,WCK11/12,WCK12/
13,WCK13/14,WCK14/15,WCK15/16,WCK16/
17,WCK17/18,WCK18/19,WCK19/20,WCK20/
21,WCK21/22,WCK22/23,WCK23/24,WCK24/
25,WCK25/26,WCK26/27,WCK27/28,WCK28/
29,WCK29/30,WCK30/31,WCK31/32,WCK32/
*
```

```
  INITIAL XH$BUSRQ,0
  INITIAL XH$COLL,0
 *
  GENERATE 10,FN$EXP,,,,4PB,4PF          GENERATE MESSAGE
REQUESTS
  GATE LR STA1                           SEIZE STATION 1
  ASSIGN 1,1,PB                          GET STATION NUMBER
  TRANSFER ,XMIT                         TRY TO SEND A MESSAGE
 *
  GENERATE 10,FN$EXP,,,,4PB,4PF          GENERATE MESSAGE
REQUESTS
  GATE LR STA2                           SEIZE STATION 2
  ASSIGN 1,2,PB                          GET STATION NUMBER
  TRANSFER ,XMIT                         TRY TO SEND A MESSAGE
 *
  GENERATE 10,FN$EXP,,,,4PB,4PF          GENERATE MESSAGE
REQUESTS
  GATE LR STA3                           SEIZE STATION 3
  ASSIGN 1,3,PB                          GET STATION NUMBER
  TRANSFER ,XMIT                         TRY TO SEND A MESSAGE
 *
  GENERATE 10,FN$EXP,,,,4PB,4PF          GENERATE MESSAGE
REQUESTS
  GATE LR STA4                           SEIZE STATION 4
  ASSIGN 1,4,PB                          GET STATION NUMBER
 *
 XMIT ASSIGN 1,FN$MAP,PF                 MAP STATION ID TO CPU
FACILITY
  LOGIC S PF1                            AND MARK FACILITY OFF
LIMITS
  ASSIGN 2,0,PB                          TURN OFF ACK BIT
 *
 DEST ASSIGN 3,V$DST,PB                  GET DESTINATION
  TEST NE PB1,PB3,DEST                   MAKE SURE DESTINATION IS
NOT SOURCE
  ASSIGN 4,PB1,PB                        SAVE STATION ID
  ASSIGN 1,PB3,PB                        MAKE STATION ID DEST ID
  ASSIGN 3,FN$MAP,PF                     MAP DEST ID TO DEST CPU
  ASSIGN 1,PB4,PB                        RESTORE STATION ID
  ASSIGN 4,FN$WCN,PF                     MAP STATION ID TO WAIT
CHAIN
 *
 DLY2 GATE LR BUS                        WAIT FOR BUS TO BE FREE
 DLY4 ADVANCE FN$DLY                     CHECK IF BUSY
  TEST E XH$BUSRQ,0,DLY4                 AND WAIT IF SO
  ADVANCE 8,0                           TIME TO INDICATE BUSY
STAE
 DLY3 SAVEVALUE BUSRQ+,1,XH              INCREMENT BUS REQUEST
COUNTER
  LOGIC S BUS                            MARK BUS BUSY
 DLY5 ADVANCE 60,0                          SEND HEADER
  TEST NE XH$BUSRQ,1,TOK1                ARE WE THE ONLY ONE ON?
  SAVEVALUE COLL+,1,XH                   INCREMENT COLLISION
COUNTER
```

```
  ADVANCE FN$DLY                    ENTER RANDOM DELAY
  TEST E XH$BUSRQ,XH$COLL,DLY2      HAVE ALL COLLISIONS BEEN
TRAPPED?
  SAVEVALUE BUSRQ,0,XH              YUP - ZERO BUS REQUEST
  SAVEVALUE COLL,0,XH                   AND COLLISION
COUNTER
  LOGIC R BUS
  TRANSFER ,DLY2                    AND TRY AGAIN
 TOK1 ADVANCE FN$LEN                SEND THE MESSAGE
  SAVEVALUE BUSRQ-,1,XH             MARK US OFF THE BUS
REQUEST
  LOGIC R BUS
  TEST E PB2,1,WCK                  WAIT FOR AN ACK
  UNLINK E PF4,RACK,1,2PF,PF2             LET THE SOURCE
CONTINUE
  LOGIC R PF1                       RELEASE THE NETWORK
 *TABULATE TACK                     TABLE ACK TRANSIT TIME
  TERMINATE 0                       ACK HAS BEEN PROCESSED
 *
 WCK TRANSFER BOTH,,BUSY            SEE IF PACKET WAS
RECEIVED
  GATE LR PF3                       TRY TO SEIZE REMOTE CPU
  LOGIC S PF3
  MARK 2PF                          MARK CURRENT CLOCK TIME
  SPLIT 1,SACK                      MAKE A COPY AT SACK
  SPLIT 1,MON                       MAKE A MONITOR COPY
  LINK PF4,FIFO                     PLACE TASK ON CHAIN
 *
 SACK ASSIGN 1,PB3,PB               MAKE SOURCE DESTINATION
  ASSIGN 1,FN$MAP,PF                MAP STATION ID TO CPU
  ASSIGN 2,1,PB                     MAKE THIS AN ACK PACKET
  ADVANCE 100                       TURNAROUND DELAY
  TRANSFER ,DLY2                    AND SEND ACK
 *
 MON ADVANCE XH$RST                 WAIT FOR THE ACK TO COME
BACK
  UNLINK E PF4,DLY2,1,2PF,PF2             RETRY IF WE TIMED
OUT
  TERMINATE 0                       DESTROY MONITOR COPY
 *
 RACK LOGIC R PF1                   WE GOT OUR ACK, RELEASE
CPU
 *TABULATE TMES                     TABULATE MESSAGE TRANSIT
TIME
  SAVEVALUE XB$IDX,V$UTL,XF
  TERMINATE 1                       AND TERMINATE
 *
 BUSY ADVANCE XH$RST                STRAIGHT TIMEOUT
  TRANSFER ,DLY2                    AND TRY AGAIN
 *
 DST VARIABLE PB1+16
 UTL VARIABLE 10000*(XH$LEN+180)/C1
 *
       REPORT     REPT
```

```
        GRAPH       X,1,8
        ORIGIN      50,10
        X           ,5,3
        Y           0,2,50,1
        ENDGRAPH
        ENDREPORT
*
RUN STARTMACRO
 INITIAL XH$RST,=A
 INITIAL XB$IDX,1
 INITIAL XH$LEN,380
 START 100,NP
 CLEAR XH$RST,XF1-XF8
 RESET
 INITIAL XB$IDX,2
 INITIAL XH$LEN,700
 START 100,NP
 CLEAR XH$RST,XF1-XF8
 RESET
 INITIAL XB$IDX,3
 INITIAL XH$LEN,1020
 START 100,NP
 CLEAR XH$RST,XF1-XF8
 RESET
 INITIAL XB$IDX,4
 INITIAL XH$LEN,1320
 START 100,NP
 CLEAR XH$RST,XF1-XF8
 RESET
 INITIAL XB$IDX,5
 INITIAL XH$LEN,1660
 START 100,NP
 CLEAR XH$RST,XF1-XF8
 RESET
 INITIAL XB$IDX,6
 INITIAL XH$LEN,1980
 START 100,NP
 CLEAR XH$RST,XF1-XF8
 RESET
 INITIAL XB$IDX,7
 INITIAL XH$LEN,2300
 START 100,NP
 CLEAR XH$RST,XF1-XF8
 RESET
 INITIAL XB$IDX,8
 INITIAL XH$LEN,2560
 START 100,,,,REPT
 ENDMACRO
*
RUN MACRO 2500
RUN MACRO 5000
RUN MACRO 10000
RUN MACRO 20000
*
```

END