6-24-1993

Fault tolerant and integrated token ring network

Thomas Christopher Gilbar
Florida International University

FLORIDA INTERNATIONAL UNIVERSITY
Miami, Florida

Fault Tolerant and Integrated Token Ring Network

A thesis submitted in partial satisfaction of the requirements for the degree of

Master of Science in Computer Engineering

by

Thomas Christopher Gilbar

1993

To: Dean Gordon R. Hopkins
College of Engineering and Design

This thesis, written by Thomas Christopher Gilbar, and entitled, Fault Tolerant and Integrated Token Ring Networks having been approved in respect to style and intellectual content, is referred to you for judgement.

We have read this thesis and recommend that it be approved.

John C. Comfort

Malcolm L. Heimer

Kang K. Yen

Wunnava V. Subbarao, Major Professor

Date of Defense: June 24, 1993

The thesis of Thomas Christopher Gilbar is approved.

Dean Gordon R. Hopkins
College of Engineering and Design

Dr. Richard L. Campbell
Dean of Graduate Studies

Florida International University, 1993

To my family and friends

for their love and support

ACKNOWLEDGEMENTS

I would like to acknowledge the efforts of Margaret Rose Dabdoub for spending countless hours helping me put this work together. I would also like to acknowledge Isidro Alvarez and Carol Levay for lending me their expertise and advice.

My thanks to Dr's Subbarao, Comfort, Yen, and Heimer for being so patient and sharing their knowledge with me.

I am indebted to Marbeth Cochran, Lei Lani Boney, and Pat Brammer for their support and for keeping me going.

I would like to thank Mike Urucinitz, Hamid Ghassemi, and Dr. James Story for helping me get the equipment I needed.

Thank you also to Pablo Perez, Irma Fernandez, Peter Hoo, Mark Williams, and Oscar Rios for listening to my rambling and telling me what I needed to do.

Finally, I would like to thank my family for their support and love over the past 25 years.

ABSTRACT OF THE THESIS

Fault Tolerant and Integrated Token Ring Networks

by

Thomas Christopher Gilbar

Florida International University, 1993

Miami, Florida

Professor Wunnava Subbarao, Major Professor

This thesis is a study of communication protocols (token ring, FDDI, and ISDN), microcontrollers (68HC11EVB), and fault tolerance schemes. One of the major weaknesses of the token ring network is that if a single station fails, the entire system fails. A scheme involving a combination of hardware and timer interrupts in the software has been designed and implemented which deals with this risk. Software and protocols have been designed and applied to the network to reduce the chance of bit faults in communications. ISDN frame format proved to be exceptional in its capacity to carry echoed data and a large variety of tokens which could be used by the stations to test the data. By its very nature, the token ring supplied another major fault detection device by allowing the data to be returned and tested at its source. The resulting network was successful.

TABLE OF CONTENTS

List of Figures:

List of Tables:

List of Symbols:

BRI	Basic Rate Interface
FDDI	Fiber Distributed Data Interface
IEEE	Institute of Electrical and Electronics Engineers
ISDN	Integrated Services Digital Network
PRI	Primary Rate Interface

ABSTRACT OF THE THESIS

Fault Tolerant and Integrated Token Ring Networks

by

Thomas Christopher Gilbar

Florida International University, 1993

Miami, Florida

Professor Wunnava Subbarao, Major Professor

This thesis is a study of communication protocols (token ring, FDDI, and ISDN), microcontrollers (68HC11EVB), and fault tolerance schemes. One of the major weaknesses of the token ring network is that if a single station fails, the entire system fails. A scheme involving a combination of hardware and timer interrupts in the software has been designed and implemented which deals with this risk. Software and protocols have been designed and applied to the network to reduce the chance of bit faults in communications. ISDN frame format proved to be exceptional in its capacity to carry echoed data and a large variety of tokens which could be used by the stations to test the data. By its very nature, the token ring supplied another major fault detection device by allowing the data to be returned and tested at its source. The resulting network was successful.

Chapter 1

Introduction

1.0 A Brief History of Microcomputers, Networks, and Fault Tolerance:

1.0.1 Development of the Computer

Computers, or things that can be loosely defined as a computer, have been around for over a hundred years. Changes in semiconductor technology in the past three decades have allowed the development first of small scale integration (SSI), then medium scale integration (MSI), large scale integration (LSI), and finally very large scale integration (VLSI). These developments have allowed more and more transistors, diodes, and resistors to be placed on a single chip, thus giving rise to microcomputers. A simple lap top computer now has more computing power than a computer which would have taken up an entire room in the 1950's.

Intel Corporation developed a 4-bit programmable device called the Intel 4004 using the new semiconductor technology. This microprocessor was replaced with the Intel 8008, an 8-bit processor, which was, in turn, replaced with the Intel 8080. The Intel 8080 was used in control applications and as the CPU in small computers in the mid-1970's. This was the advent of what is now referred to as the microcomputer.

Motorola followed the example of the Intel 8080 by releasing the 6800 a few years later, a microprocessor designed with a different architecture and instruction set than its Intel equivalent, the 8085. Both the 8085 and the 6800 were vastly superior to the 8080 in computing power. These new microprocessors were no longer used as simple programmable logic devices, they were now considered CPU's.

The VLSI technology led to the development of even more powerful microprocessors. Intel and Motorola developed 16-bit, 32-bit, and now even 64- bit microprocessors. These developments led to the single board microcomputer, which in turn led to cheaper microcomputers which were accessible by almost any person who desired to have a computer. Motorola released the MC68HC11, a powerful 2 MHz, 8-bit microcomputer with advanced on chip peripheral capabilities. Later Motorola released a microcomputer with the MC68020 for a CPU. The 68020 is a 33MHz, 32-bit microprocessor.

In the mid 1970's, IBM developed the 801 minicomputer, a computer with a microprocessor whose main function is to reduce the data path cycle time, regardless of whatever else it may need to sacrifice. This was the first of what is called Reduced Instruction Set Computers (RISC) processors. RISC processors are known for their speed. To achieve high speeds during processing, they have several special characteristics. First, they have simplified instructions and addressing modes. The RISC processors have a reduced instruction set, which results in some of the more basic, and most common, processor functions, such as loads and stores, being optimized. Next, multiple execution units allow parallel processing. The RISC machines also have a pipelined architecture. Finally, the RISC processors utilize a larger set of registers to minimize the usually time consuming memory accesses. However, with the reduced instruction set comes the need for compilers which are more complex.

Another type of processor recently developed is called the Digital Signal Processor. These devices lean more toward speeding up arithmatic functions rather than reducing data input/output time. The faster computation insures that the processing rate is limited only by the speed of data transfer between the processor and memory. DSP's have been optimized for high speed execution of signal processing functions such as dot products and Fast Fourier Transforms. To achieve the faster arithmetic, the DSP's include fast hardware multipliers and totally independent data and program storage areas. They also have fast interrupt switches for more efficient task switching.

Multiple ports allow for the design of efficient parallel processing architectures. A steady flow of data will allow a DSP to run as fast as a RISC machine. However, any break in the data path can result in performance loss.

1.0.2 Networks

With the proliferation of computers came the need to allow communications between computers. Users needed to share software, data, and other forms of information, whether it was between computers on opposite sides of the world or between computers on the same desk.

No doubt the ideal way to communicate between computers would be to supply communications and control lines from every computer to every other computer in the world. However, setting up these dedicated lines would be expensive, time consuming, and would take up too much room. For example, stringing a dedicated line between two computers which are several thousand miles apart would be very expensive. Also, most computers require access to peripherals such as printers, hard drives outside of their own computers, modems, etc. However, these peripherals are expensive. If the user only needs to use these peripherals occasionally, the expense may be excessive.

A solution to this problem is to attach a computer to a network, so that while a user may not have a direct connection to a needed device, s/he can establish a connection by one of two ways:

1. First, by going through another computer on the same network that may have a direct connection with the device. For example, Figure 1.1 at the end of this section shows station 1 passing through station 2 to gain access to its printer.

2. By "borrowing" a line shared by several computers which accesses the device. In the case illustrated in Figure 1.2 (again at the end of this section), station 1 is accessing a printer by taking control of a shared line.

These solutions lead to other problems, however. In either case, the computers must share a common connection. The single line cannot handle more than one communication at a time. A scheme must be developed to decide who controls the line at any given moment. Further, each station on the network deserves its fair share of access time to the network.

In the 1970's and 1980's, computer science and data communications combined to come up with networking schemes.

Protocols and computer communication architectures were designed to meet the needs of various users. Ring, bus, and star topology networks were developed.

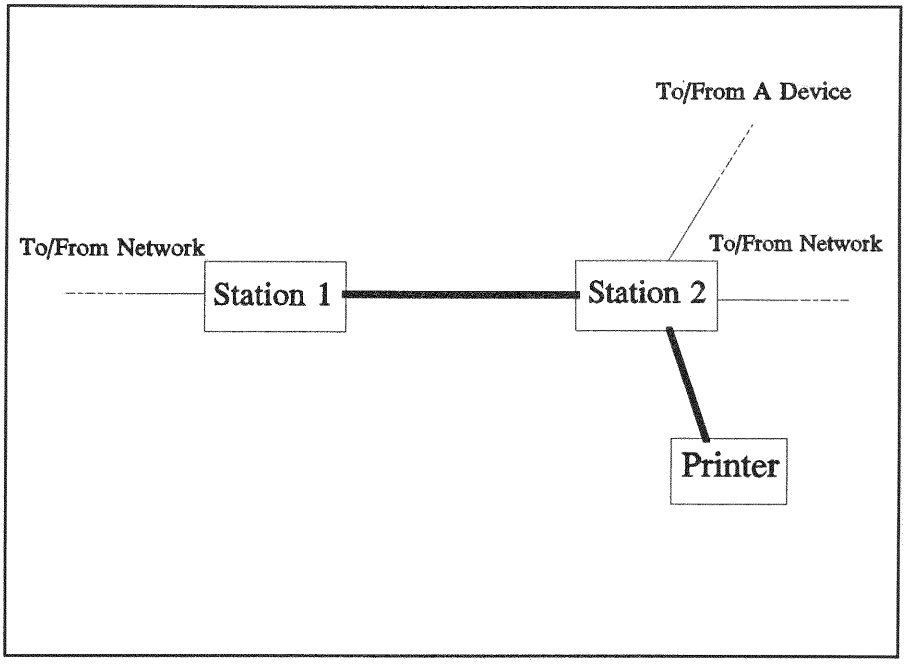

Figure 1.1: Computer Accessing a Printer Through Another Computer

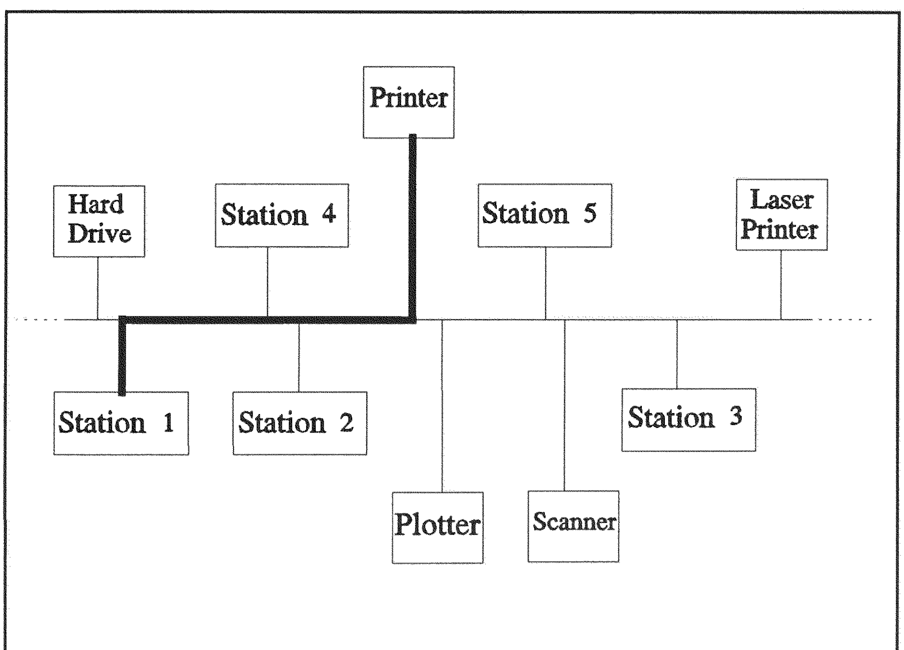

Figure 1.2: Computer Accessing a Printer By Seizing Control of a Common Line

1.0.3 Token Ring

One scheme is the token ring network. Proposed in 1969, this is the oldest of the ring network schemes. The concept of the token ring is relatively simple: A token, a small packet of information different from any data that might be sent, is placed on the network. Only one of these tokens may be present on the network at a given time. If a station wishes to send data, it must wait for the token to arrive at its node on the network. The station removes the token from the network and sends its message. By removing the token from the network, no other station can receive the token, and therefore no other station can send data. Once the station finishes with its data transmission, it places the token back on the ring, potentially enabling further communications. By forcing a station to wait for the token before it can send a message, the token ring network removes the possibility of data corruption and collision. A more detailed description of the token ring will come later in this document.

1.0.4 Fault Tolerance

One major problem in any network is ensuring that the information received is exactly the information sent. Any number of problems can occur in a system. The wrong address can be supplied, thus sending the data to the wrong place. Both the sender and the receiver need to have ways to find this out. The data may get scrambled while it is on the line. One of the stations on the network may go down while the data is being sent, causing part or all of the message to be lost.

The solution to these and other problems is to make the networks fault tolerant. In other words, there must be a way to detect and correct errors in transmission. Several methods, or combinations of methods, may be used: Coding (Huffman, Hamming, parity generation and checking, etc), hardware modular redundancy, acknowledgement signals, etc. Several methods will be reviewed later in this document.

1.1 Introduction to this document

The main objective of this thesis is to develop a communications network which improves a popular LAN protocol by making it more fault tolerant and adapting it to modern technology. The token ring network is the basic protocol that will be used in this work. This topology was chosen for several of its characteristics, including that it is one of the three major LAN's, the innate security which comes from its circular layout, and the difficulty of creating a token ring network which does not completely fail when any of its stations fail. To develop this network, a variety of software and hardware techniques will be altered and combined. The result should be a more fault tolerant network which has been altered to compatible with modern protocols.

As stated above, the advent of computers led to the need to communicate between computers as well as computer peripherals. Simply connecting every computer to every other computer and to its own peripherals is too expensive and unrealistic. Therefore, fast, efficient networks must be developed and implemented.

This document will show the step by step process of designing one such network: The Token Ring. Microcontrollers such as the motorola MC68HC11 will be used for modeling and testing the system. The ring designed can consist of up to 256 stations (256 distinct addresses)(see Figure 1.3 on the following page). The actual model developed will have three such stations. IBM PC's will be used to emulate the stations. Programming for the HC11's will be in assembly language for efficiency and speed of operation.

Simply building a token ring network is not enough. The token ring network must also be highly fault tolerant. This will be achieved through a variety of methods: address testing, encoding schemes, acknowledgements, etc. This document will focus on the best way or combination of ways to create a fault tolerant token ring network. Criteria for choosing the best way will include speed, efficiency, difficulty in installing, and of course, ultimate reliability. The concepts here can be extended to major fault tolerant data and computer communications.

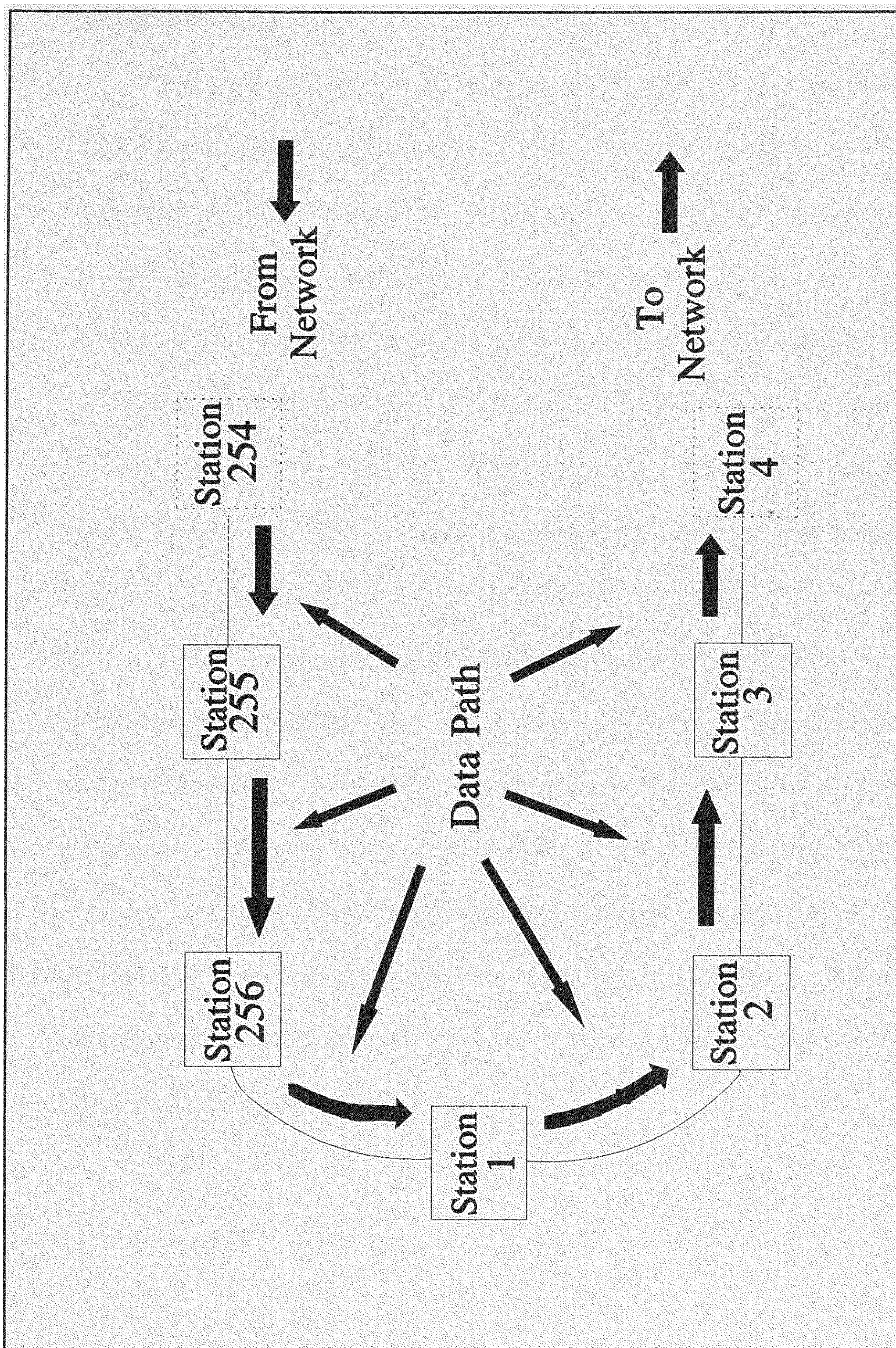

Figure 1.3: 256 Station Token Ring Network

1.2 Chapter Organization

This document will be divided into 8 chapters and two appendices. Following the introduction, Chapter 2 will contain a detailed look at the considerations in the design of the system, which will include a discussion of the token ring network concepts and design and fault tolerance techniques. Chapter 3 will contain a discussion of the hardware used in this research. This will include a description of the 68HC11 board, the IBM PC's, and how the network was developed with the microcontrollers. Chapter 4 will be a discussion of the system integration techniques, including protocols and methods. Chapter 5 will be a description of the software developed for this project. It will include a discussion of the particular needs of software for the token ring network (including fault tolerance) and the software needed to achieve communications between the various hardware elements of the system. Chapter 6 will show a variety of applications for the token ring network that will be developed. Chapter 7 will be the conclusions and will discuss areas for future study and experimentation as well as design alternatives and system enhancements. Diagrams and the software source code routines will be including in the appendices.

Chapter 2

Design Concepts for a Token Ring Network

2.0 Introduction

Chapter 2 presents a study of the concepts used to develop the network for this thesis. Sections 2.1 through 2.4 of this chapter discuss the token ring network. Topics include the basic concept of token ring networks, official CCITT protocols, factors and methods used to design the token ring, and a comparison of the token ring network to other types of networks. Sections 2.5 and 2.6 of this chapter discuss fault tolerance. First, the basic concepts of fault tolerance will be discussed, including what it is, why it is necessary, and some basic fault tolerant schemes. Next, various fault tolerant schemes that could be used in this project will be discussed in detail.

2.1 Concept of Ring Networks

2.1.1 Topologies

Networking computers has created some problems for designers. Directly connecting two or three computers is a relatively easy matter. Connecting four computers is a little more difficult, but still can be done. However, when the number of terminals increases to five or more, there are just too many connections to be made. Also, with more connections, more intelligence is required. Each terminal needs to know exactly which connection leads to which terminal and needs to be able to listen in to all of the lines to see if data is being sent to its location. Intelligence and massive amounts of connections are both time consuming and expensive.

For these reasons, several networking topologies have been developed over the past couple of decades. The majority of the schemes fall under three basic topologies: The Star, Tree, and Ring. A special type of Tree, called a Bus topology, is also used quite often.

As seen in Fig. 2.1, the terminals in the star network communicate through a switching element. This element establishes a dedicated path between the receiving terminal and the transmitting terminal. The star networks are easy to expand and require very few connections (each terminal has only one network connection, which is to the central node). Most of the intelligence is based in the central element, leaving very little for the terminals to do when it comes to networking. Since each terminal has its own line, the bandwidth requirements of the lines are quite small. The most difficult part of developing this topology is in developing ways of preventing data collisions. Data collisions occur when two or more terminals simultaneously attempt to send a message on the same line. The messages will meet and the result will be the loss of data and the possible destruction of all messages. Two or more terminals can not simultaneously send data to a third terminal without risking the loss of some or all of the information. If anything but a full duplex line (a line which can send data in both directions concurrently) is being used, a terminal cannot transmit at the same time it is receiving. The more transmitters which are allowed to send simultaneously, the more intelligence and memory is required in the central element. Acknowledgement, request to send, and/or time division multiplexing schemes can be developed to alleviate these problems. A major weakness for this topology is the dependance on the central node. If it fails, the entire system will fail along with it.

Figure 2.2 shows a typical Tree topology. The bus topology is similar, but it does not have branches. Instead, it has one single "trunk" on which all of the stations are connected. For these topologies, a multi-point medium must be used. That is, several stations share the same transmission medium. Unfortunately, with only one communications medium, only two terminals can communicate at a time. Data collisions are difficult to avoid. A transmitting station must seize control of the line to keep other stations from attempting to communicate simultaneously. Again, acknowledgement and request-to-send schemes must be developed. The required number of connections is minimized. Intelligence is located at the terminals; however, since each terminal has to watch only one line, the required intelligence is less than if every terminal was connected to every other terminal.

The topology which is the most important to this thesis is the Ring (Figure 2.3). Terminals are connected to the network through a repeating element. The network itself is a closed loop. This increases security by making it difficult for an unauthorized station to break into the network. Data circulate through the network using connections between the repeaters until it reaches the receiver. The receiver makes a copy of the data, and continues to forward it. The transmitter uses the returned message as a form of acknowledgement. When the message returns to the source, the transmitter removes it from the network. Control of the network is then sent on to another station. Problems with this type of topology include setting up the order of the stations for sending messages, and removing a packet from circulating through the network, especially in the case of errors in addressing. These problems are solved by creating some type of control protocol. Another potential problem with the ring topology is that any break in the ring, or a failure of a repeater, will cause a failure in the whole network. Also, adding another terminal to a ring network can be difficult because it involves finding two close stations that are connected, breaking their connection, and then connecting each of them to the new station. Changes must be made in the basic ring design to deal with these problems.

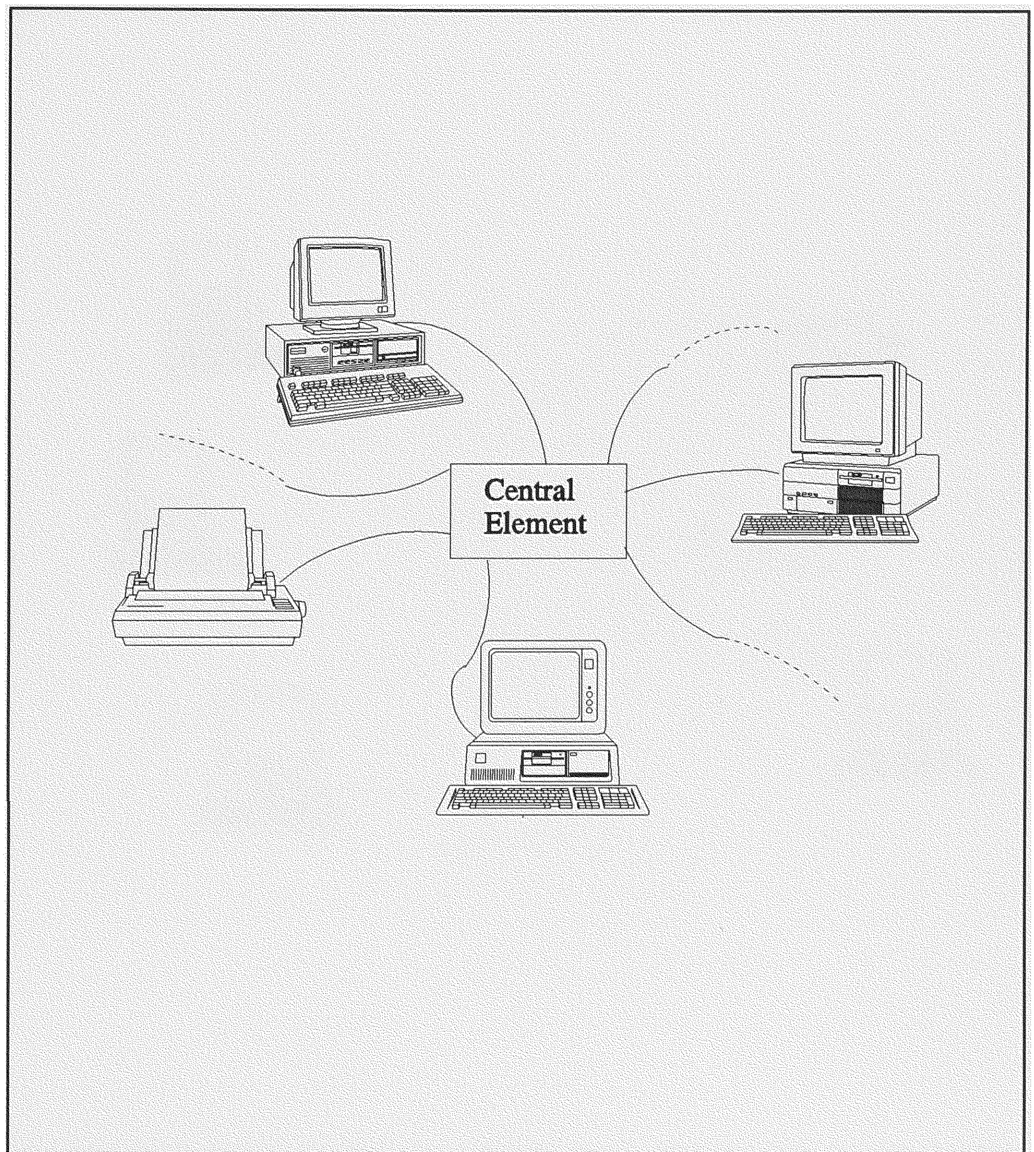

Figure 2.1 Example of a Star Topology

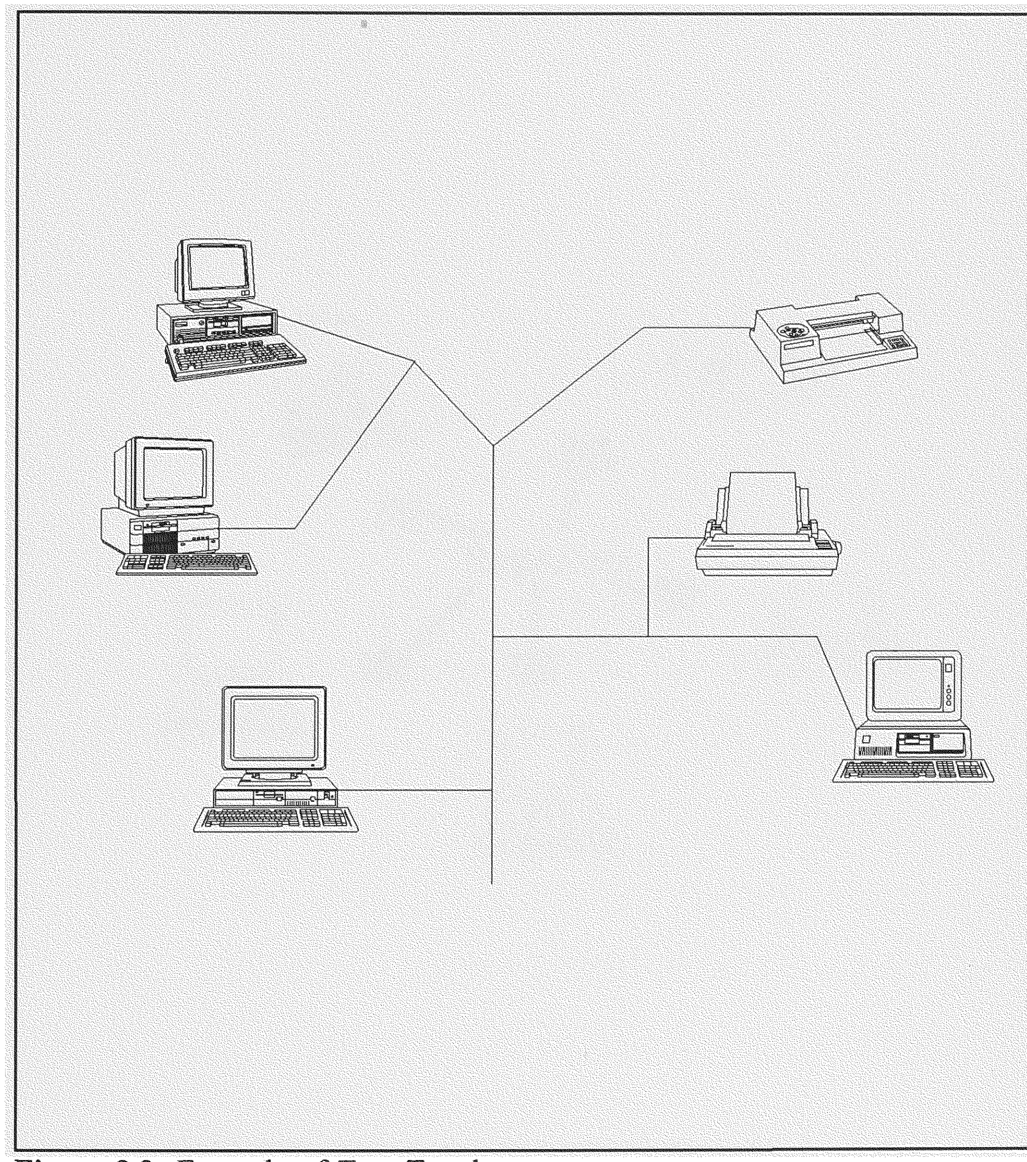

Figure 2.2 Example of Tree Topology

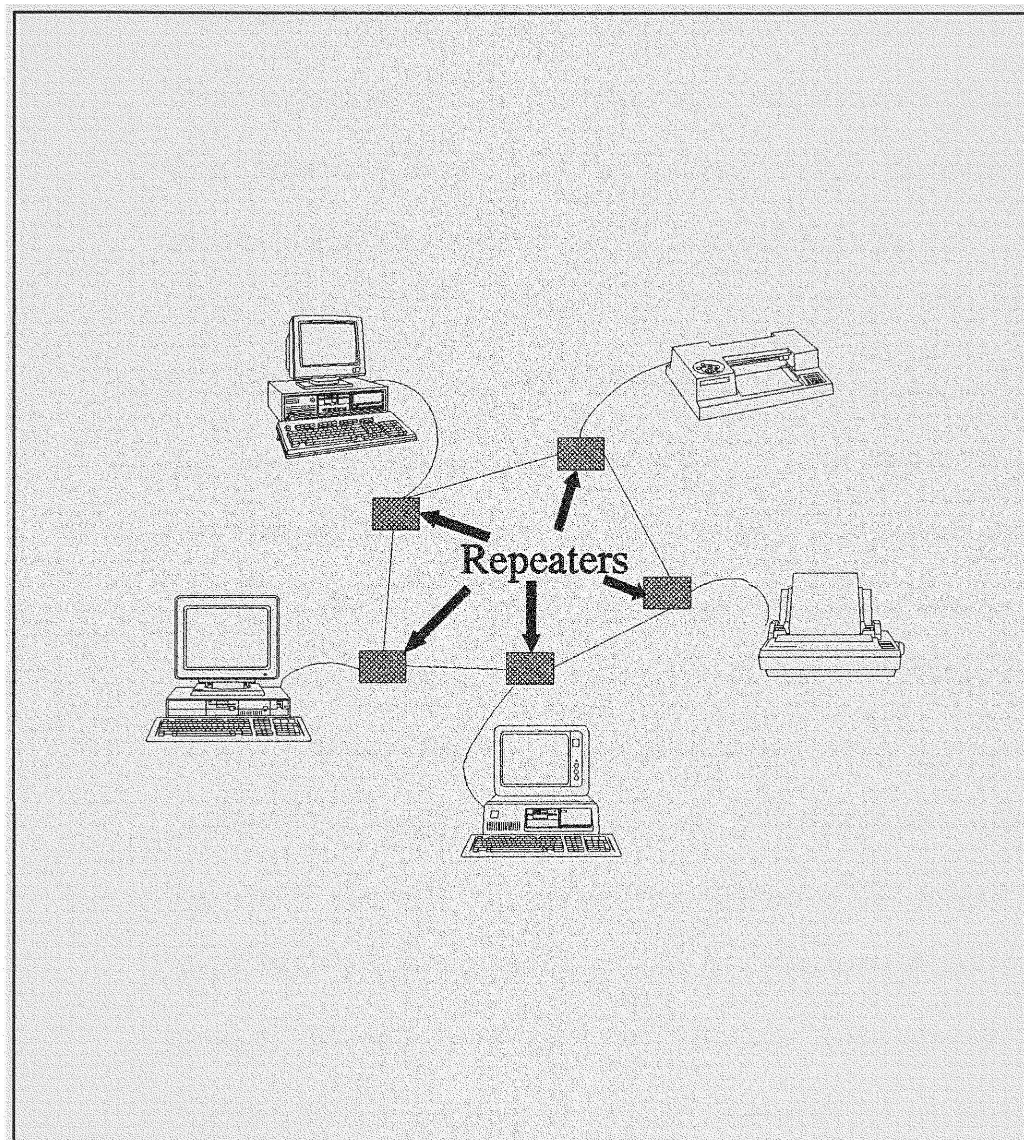

Figure 2.3 Example of a Ring Topology

2.1.2 The Ring Repeater

The intelligence for this topology is mainly in the repeaters. For this thesis, the Motorola 68HC11's will be used as the repeaters. These will be used for data encoding, storing data until it can be sent, error detection/correction, etc. Notice that any similar microcontroller, such as the Intel 8051, could be used with similar results. The reasons for choosing the 68HC11 will be outlined later in this chapter.

Repeaters have three possible states: Listening, Transmitting, and Bypassing. In the listening state (Fig 2.4), the repeater receives data from the network, copies it, then puts it back on the network. This creates a slight delay. At the beginning of a message, the repeater tests the destination address. If the repeater detects its own address, it remains in the listening state. If the destination address does not belong to the terminal, there is no longer any need for the repeater to copy the data for the station. It will transfer into the bypass state (Figure 2.5). In this state, the data is passed directly on to the next station on the network, removing the delay. The final state is the transmit state (Figure 2.6). In this state, the repeater listens in one direction and sends in the other. The listening side is picking up the message that the transmitting side is sending, but with at least a one bit delay (the ammount of delay depends on the transmission flow). This serves as an acknowledgement. Sometimes certain bits are modified by the receiving station to show that the packet was, indeed, copied.

2.1.3 Brief Introduction to the Token Ring

One control protocol is called the token ring. This technique was first proposed in 1969 (it was called the Newhall ring at the time), making it the oldest ring protocol. This protocol was developed to remove the risk of data collisions, increase ring efficiency, and to supply a means of determining the order in which the stations will be allowed to access the network. In the next section, a detailed description of how this is done is presented.

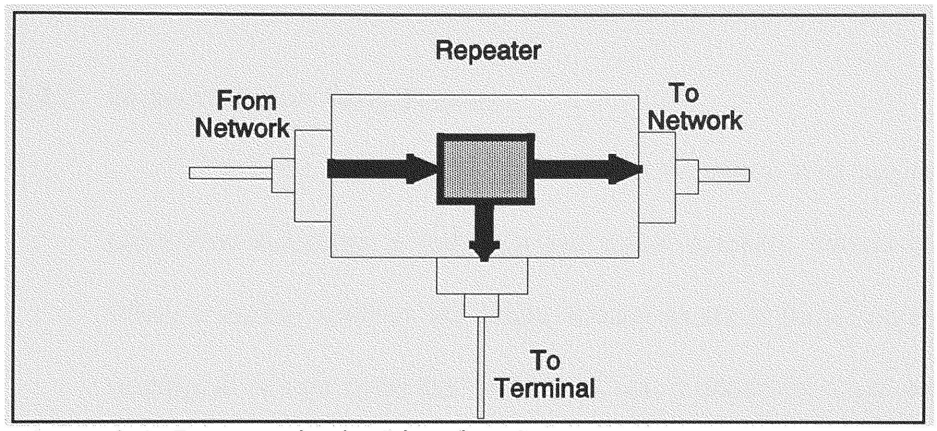

Figure 2.4 Repeater in the Listening State

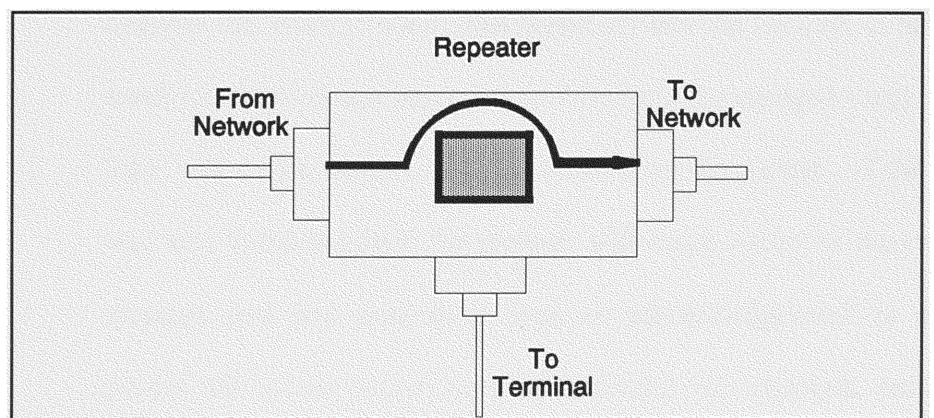

Figure 2.5 Repeater in the Bypass State

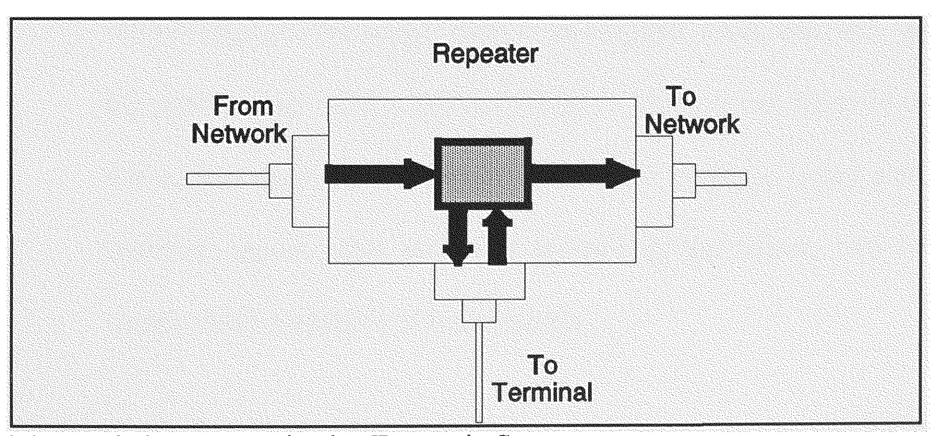

Figure 2.6 Repeater in the Transmit State

2.2 Token Ring Protocols

2.2.1 General Token Ring Protocol

To solve some of the problems of a general ring network, the token ring circulates a small packet, called a token, around the ring. When a station receives the token, it tests to see if there is a message waiting to be sent from that station. If not, it sends the token on to the next station on the ring (Figure 2.7). If there is a message, the station changes the token to warn other terminals that the network is busy, and sends out the message immediately following the changed token (Figure 2.8). All of the stations test the address on the packet. If the packet was sent to the terminal, the repeater will make a copy of the complete message and will send some type of acknowledgement (a separate packet, the whole packet itself with certain bits changed, part of the packet, etc.). Once the source has received an acknowledgement, the station changes the token back to its original form, and will send it to the next station on the network (Figure 2.9). Since there is only one token, it is only on the ring if there is no message being sent, and a terminal cannot send unless it is holding the token, two terminals cannot send messages simultaneously.

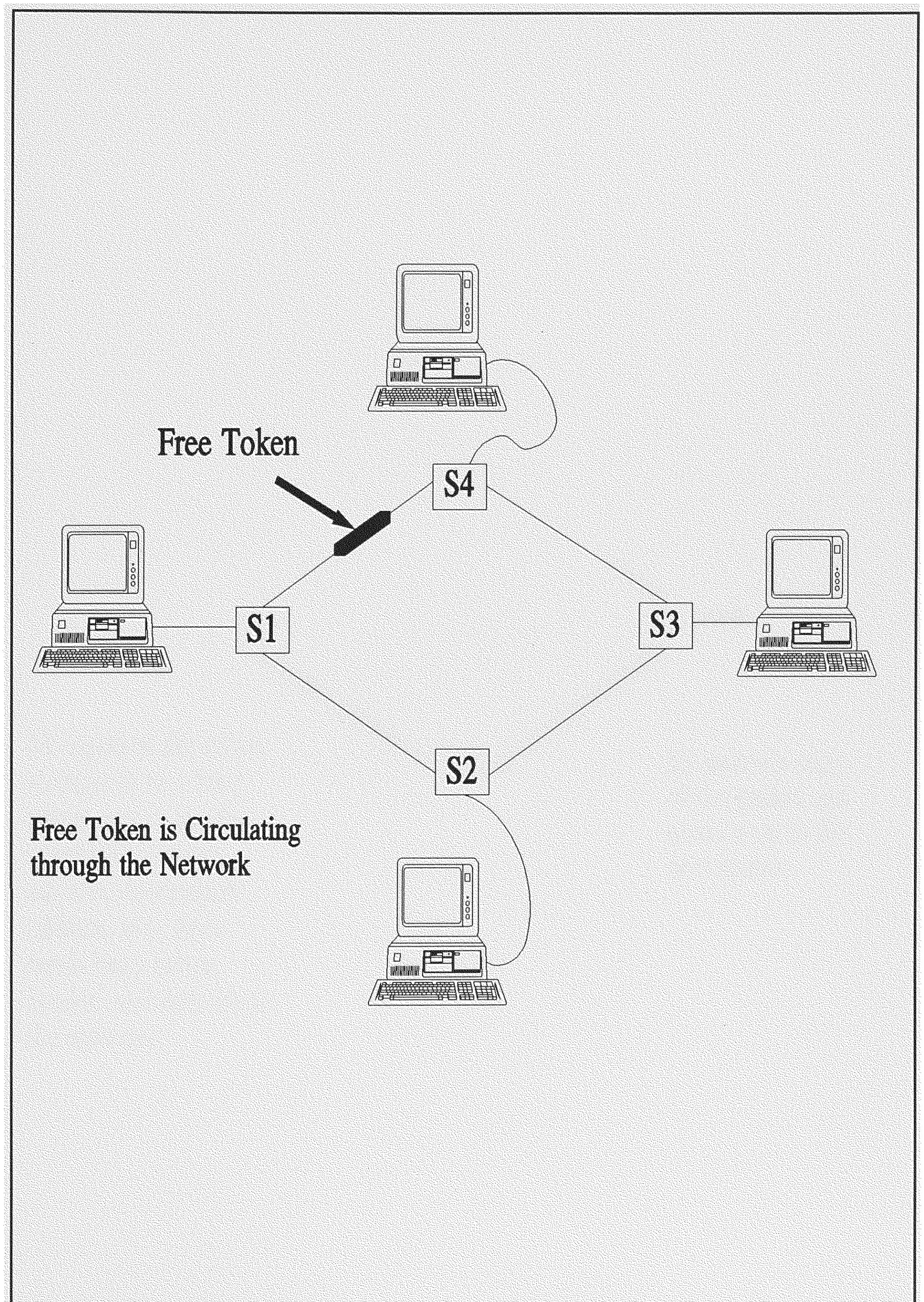

Figure 2.7 Free Token Circulates on a Token Ring Network

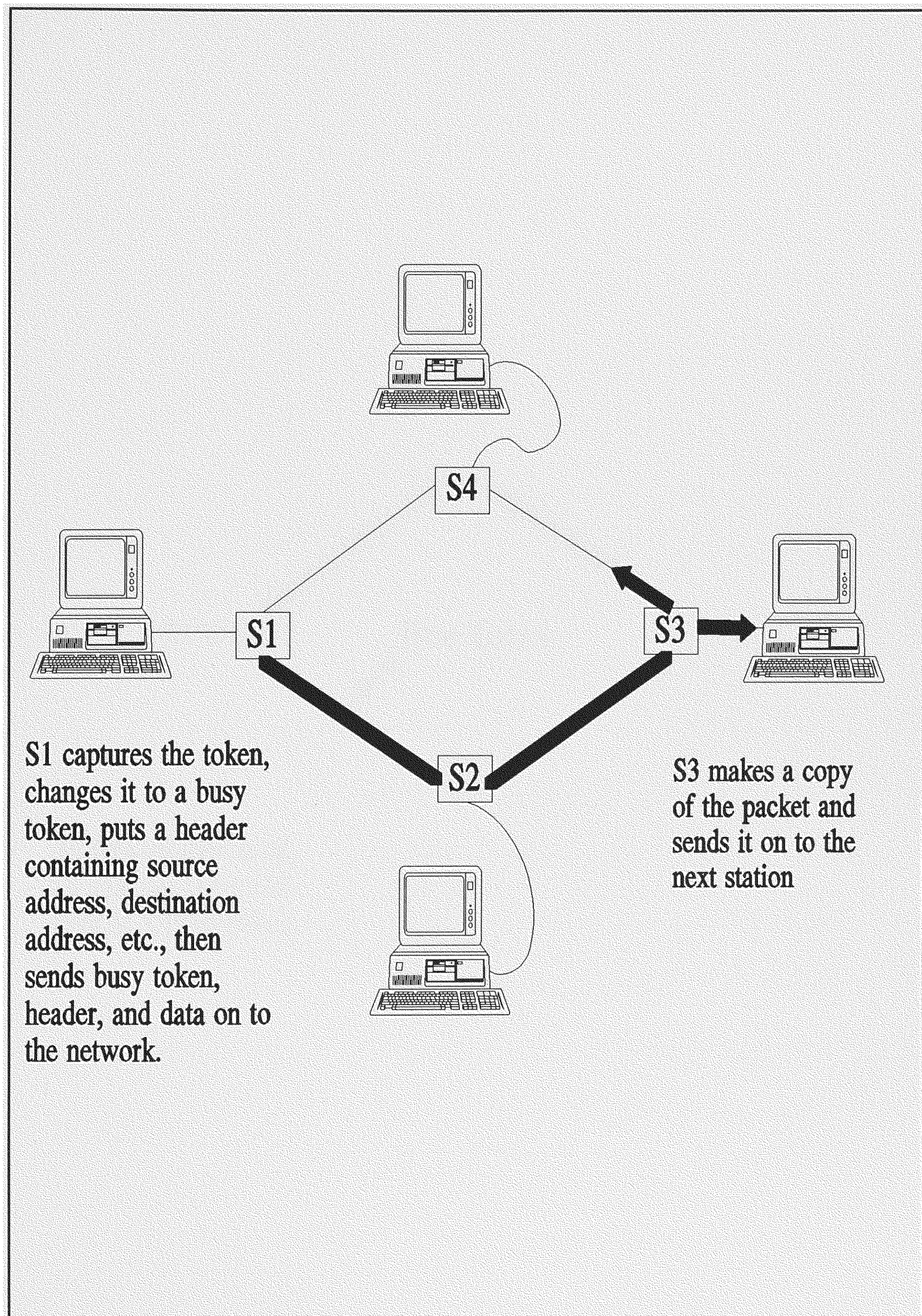

Figure 2.8 S1 Captures the Token, then Sends a Packet to S3

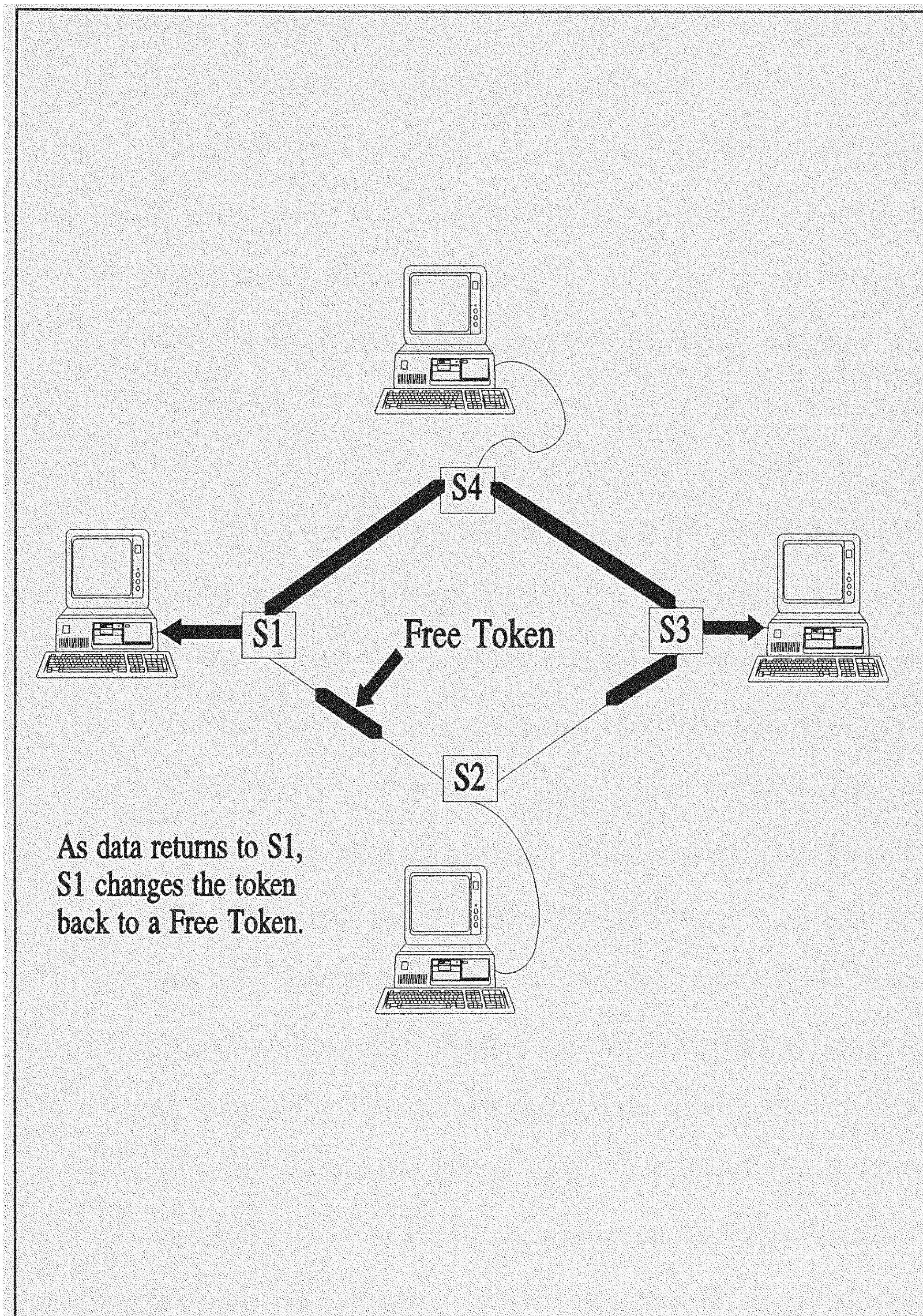

Figure 2.9 S1 Places the Free Token Back on the Network After It Receives the Acknowledgement from S3

2.2.2 Priority Schemes

There are several priority schemes possible for the token ring. The easiest, of course, is to allow each station to send out one packet of a specified size or smaller, then pass the token on to the next station in the ring. This ensures that all of the stations have equal access to the ring. However, quite often some type of priority scheme is desired.

One such priority scheme is the IEEE 802 Ring LAN Standard. For this standard, the token is a packet of data which includes a byte (or more) for priority and a byte (or more) used to reserve the token. A station wishing to send a message waits for a free token with a priority less than its priority. Once it gets this, it can send its message, along with a busy token. While a station is waiting for a free token, it can reserve a passing busy token by setting the reserve field to its priority. By doing this, a station with a lower priority cannot take a free token before the station with a higher priority. If the reserve field has already been set to some value, the station tests that value and compares it to its priority. If the priority in the reserve field is less than its priority, the station clears the old priority and sets the reserve to its priority. However, if it is higher, it cannot set the reserve field and must wait for the higher priority station to send its message. When the busy token returns to the source, the source will clear out the priority bit and then sends the now free token back on the network. The free token will circle the network until a station with a priority equal to or higher than the reserve priority picks it up, clears the reserve field, and sends a message. If the token is not reserved, any station can grab it and send a message. See Chapter 4 for more details on this protocol.

Another standard for prioritizing packets is the Fiber Distributed Data Interface (FDDI) Ring High Speed LAN standard. This method is very similar to the IEEE standard, but it is designed to utilize the high speed of the fiber more efficiently. The FDDI uses a timing algorithm which allows the stations to send larger packets. This algorithm takes into account the length of time that the station had to wait before it received the free token. With this method, the source is restricted by a time frame rather than a packet length. This allows larger, or more, packets to be sent, increasing the efficiency of the network.

2.3 Design of Token Ring System

There are several major considerations when designing a token ring network. A major one of these is fault tolerance. The biggest stumbling block for token ring, at least with the software, is improper management of the token. If a station does not return the free token to the network, or does not return ANY token to the network, the result can be disastrous. Methods for avoiding this problem are discussed later. Fault tolerance on the hardware side is more difficult, but must be considered.

Other considerations include choosing an appropriate protocol to follow. For example, there are several priority schemes that can be followed. The designer must decide which is the closest to the design specifications. The designer needs to know if one station has higher priority than the others, or if certain types of messages have higher priority than others. If priority is necessary, the best way for it to be implemented while still giving fair access to other stations must be investigated.

Communication protocols must be chosen. Frame format, encoding techniques, etc. must all be chosen and standardized throughout the network. Data rates must be chosen. All of these will be discussed in Chapter 4.

Hardware is another important factor. The design of the stations must be set. The functions of the repeaters, and the hardware used for those repeaters, must be chosen. Software must be written so that a large variety of hardware can access the network using it. For example, a good network should be able to handle both the Motorola and the Intel families of microprocessors, as well as allowing printers, plotters, etc. to be accessed by any of the stations.

Still another consideration is the structure of the token ring itself. The traditional token ring consists of a single, unidirectional path. Newer token rings, however, are bidirectional, or have more than one ring. The second direction or line can be used to choose the shortest path from the source to the destination, or for acknowledgements. Hardware and software limitations come into play when choosing the basic design of the token ring.

The specifics for the token ring which will be built for this project will be discussed in later sections and/or chapters.

2.4 Comparison of Token Ring to Other Networks

There are many other types of networks. This section will describe just a few of those to help outline the strength and weaknesses of the token ring network.

To begin with, consider the token bus. It is very similar to the token ring in its design, so more attention will be paid to it than the other network types. One of the major differences is the connections required. A bus needs a multi-point medium which can support any of a large number of devices. For the token ring, only point-to-point connections must be considered. Each station needs only worry about the next station in the ring. This gives the designer more choices when it comes to designing the connections. Unfortunately, the point-to-point connections mean that there where will be a longer delay for a message sent on the token ring. For the token bus, the only delay that the sender needs to worry about is caused by propagation delay of the connections and distance between stations. A ring must deal with this delay, and also delays as the data is sent through all the other stations on the network. Each station adds at least one bit delay. Also, the message may have to travel a greater distance through a token ring since the token ring is unidirectional. Since all stations are connected to the same element on a token bus, the instant the message is sent it will go in all directions at once. For the token ring, if it needs to communicate with the station just before it in the network, the message will have to travel all the way around the ring.

The single direction of the token ring does have its advantages. It makes some forms of fault tolerance easier to implement than on the bus. For the bus, all stations have easy access to the same transmission line. This increases the odds of an error occurring, because two stations are trying to control the network simultaneously. The stations on the ring have such a set path and such limited access to the network that data collisions are far less frequent. Also, since each station on the token ring has two lines, one for input, and one for output, the stations can more readily send and receive simultaneously without causing problems with the network. The same is not true for the bus. Unless a full duplex line is used, only one station can transmit at a time.

Another network is the pure ALOHA technique. This technique was designed to allow widely scattered terminals to access one central mainframe. To do this, the ALOHANET has two channels: One channel from the terminals to the mainframe, and another channel from the mainframe to the terminals. When a station has a packet to be sent, it does so. This creates a free-for-all on the transmission line. The ALOHA technique is very simple, and the intelligence is minimized. However, the occurrence of data collision is astronomical compared to the token ring. Token rings are more difficult to implement, require more time, energy, and money, but the savings in fault tolerance and in transmission time due to increased throughput makes it well worth the effort.

Finally, consider the Carrier Sense Multiple Access with Collision Detection (CSMA/CD). This is the most commonly used control method used for bus topologies. This system is better than the ALOHA in that when a collision is detected, transmission is instantly terminated. This saves time in sending messages that are not going anywhere anyway. However, it still suffers from a great deal of collisions, again making it weaker than the token ring in that sense. Like the ALOHA, the CSMA/CD method is easier to implement than the token ring, and messages that actually make it through get to the destination faster.

2.5 Concept of Fault Tolerance

2.5.1 Introduction

Fault tolerance is a major concern for any system, whether it is part of a larger network or a simple terminal working on its own. Designers have been working on ways of building systems which can sustain small errors without losing the whole system, or errors can be made without losing data or passing on bad data. Many methods have been developed for both error detection and error correction. Some possible problems and a few possible fault tolerant solutions will be discussed in this section.

2.5.2 General Fault Tolerant Schemes

In more complicated systems, one piece of hardware having a fault can take the entire system off line. Sometimes diagnosing the problem is difficult, and fixing it is nearly impossible. One solution for a hardware fault is called Triple Modular Redundancy (TMR). This fault correction technique involves running three identical units simultaneously in parallel. The output of the units are then sent to a voter, and the voter outputs the majority decision. Some added hardware can also supply a warning that one of the units has become faulty so that the user is aware of the problem. This system has two benefits: first, the system will not automatically go down, giving the user time to get in and fix the problem while the system is still up and running. Second, the odds of two units having simultaneous faults are much smaller than of any single unit having a fault; therefore, the efficiency of a system increases as a whole. For more important pieces of a system, five, or even seven, redundant systems can be running simultaneously, further bringing down the odds of a fault occurring. The one major weakness of redundancy lies in the cost and the space taken up by adding the extra hardware. The importance of the unit and the cost of redundancy must be balanced before action is taken.

Another major concern for network faults is in accuracy of the transmission. In the time between the source sending the data and the destination receiving the data, many things can occur to cause errors to appear in the data. Methods for the destination to test data and to inform the source that the right data was received (or even that ANY data was received) must be developed. First, ways of checking (and possibly correcting) faulty data will be discussed. A second layer of TMR can be added to the output of three voters to deal with faults in the voter.

Parity generation and detection is probably the most popular error detection/correction method. In this method, the bits in a field of data (a byte, word, long word, etc.) are compared and used to generate an extra bit of data which is then added to the field. At the receiving end, the parity bit is regenerated using the data. If it matches the parity bit sent, the destination can be certain that at least a single bit error has not occurred. Unfortunately, a single parity bit can only detect a single error in the field. To increase the chances of detecting an error, multiple parity bits can be added by comparing certain bits within the field, rather than every bit with every other bit. Each added parity bit increases the chance of detecting multiple faults, and can even eventually be used for error correction. A second layer of parity can also be added. This second layer checks the other parity bits, further increasing the chance of detecting errors, this time in the parity generation itself. Of course, there is a cost. The more parity bits that need to be sent, the less information is sent in each field of data. This decreases the throughput of the communication system. Communication time is increased and more equipment is needed to create and test the parity. Again, the number of parity bits, and the benefits of error detection/correction, must be weighed against the cost. A single parity bit for a byte of data may not be used for correcting the data, but it will warn the user about single bit faults (the most common fault), and can be used to ask for a retransmission of the data. The fewest number of bits that the code words differ by is called the Hamming Distance. The more bits used in the coding, the larger the Hamming Distance possible.

For example, assume that a system needs to encode two choices, A and B, and two-bit code words are chosen. Since only two words need to be encoded, then only two combinations are necessary, say 00 and 11. The following table results:

Table 2.1: 2 Bit Hamming Code

Possible Codes	Encoded Message
00	A
01	None
10	None
11	B

Notice that the chosen combinations differ by two bits; therefore, the Hamming Distance for this coding scheme is two. That is why 00 and 11 were chosen. Any other choice of combinations would result in only a one-bit difference. Single bit errors can now be detected. If the word 01 is received, the destination address knows that an error occurred. This is not a valid code. However, the destination address does not know if the right code was 00 or 11.

By increasing the Hamming Distance, errors can be corrected. For example, if 3 bits are now used, there are eight possible combinations. The following table results:

Table 2.2: 3 Bit Hamming Code	
Possible Codes	**Encoded Message**
000	A
001	None
010	None
011	None
100	None
101	None
110	None
111	B

By choosing 000 and 111, the new Hamming Distance is three. If the receiver now receives 001, he not only knows that there has been an error, but also that the input was supposed to be 000, assuming single bit errors (111 would be a double bit error). Like parity coding, the more bits the better chance of error detection/correction, but it also takes longer to send the message thus, throughput goes down.

A major problem in any network is finding a way to indicate to the source that the data has been received, or if the appropriate person received the data. One way to do this is to come up with an acknowledgement scheme. An acknowledgement can be anything from a single bit, to a byte, to an entire packet sent by the receiver back to the source once the transmission is completed. Sometimes the entire packet is sent back to the source with a single bit changed to indicate that it has been received. This has the added benefit of sending the exact same message back to the source so that the source can test for errors in transmission itself. If there is an error, the message can be resent. Of course, the larger the acknowledgement, the more time it takes for a transmission to be completed. Again, a balance must be struck. Longer transmissions sometimes need multiple acknowledgements, requiring the source to stop for a while, wait for the acknowledge, than resume sending. This takes up even more time.

2.5.3 Fault Tolerance in Token Ring Networks

Faults can occur in many places in a ring network. In this section, one of the worst of these will be discussed: If one of the stations on the ring goes down, the entire ring can be brought to an abrupt halt. A traditional solution will be discussed, and will be compared to the solution which will actually be implemented later in this chapter.

One traditional solution to this problem is to change the architecture of the ring itself. The ring can be made into what looks like a star ring. In this case, rather than connecting each station directly to the surrounding stations, the connections can be made to a central intelligence (Figure 2.10). In other words, a station will send its data to a central station, which will than send the data to the next station on the ring, which will then send the data back to the central station, etc. Other than the central station, the network works identical to the token ring. The strength of this set up is that if a fault occurs, the central office will see it, know where it occurred, and skip over that station to send directly to the next station. Also, connection to the central intelligence allows for easier expansion, another serious problem with ring networks.

There are several weaknesses to this method. First, if the central intelligence fails, the whole system will probably also fail. Also, if any one station goes down, it may still create a delay or even temporarily disable the entire network. Finally, the single central intelligence adds delay to the network and may require a lot of extra cable to connect it to the other stations. However, unlike the pure star network, the central intelligence does not need to be very intelligent.

To help with these problems, added changes can be made. The ring can be broken down into smaller rings, and the smaller rings are connected by what is called a bridge (Figure 2.11). With this added step, if one central intelligence, or a station, goes down, the entire network will not be affected. Delay and amount of cable may be reduced. However, the bridges must be quite intelligent to deal with the switching, and more central intelligence may be required.

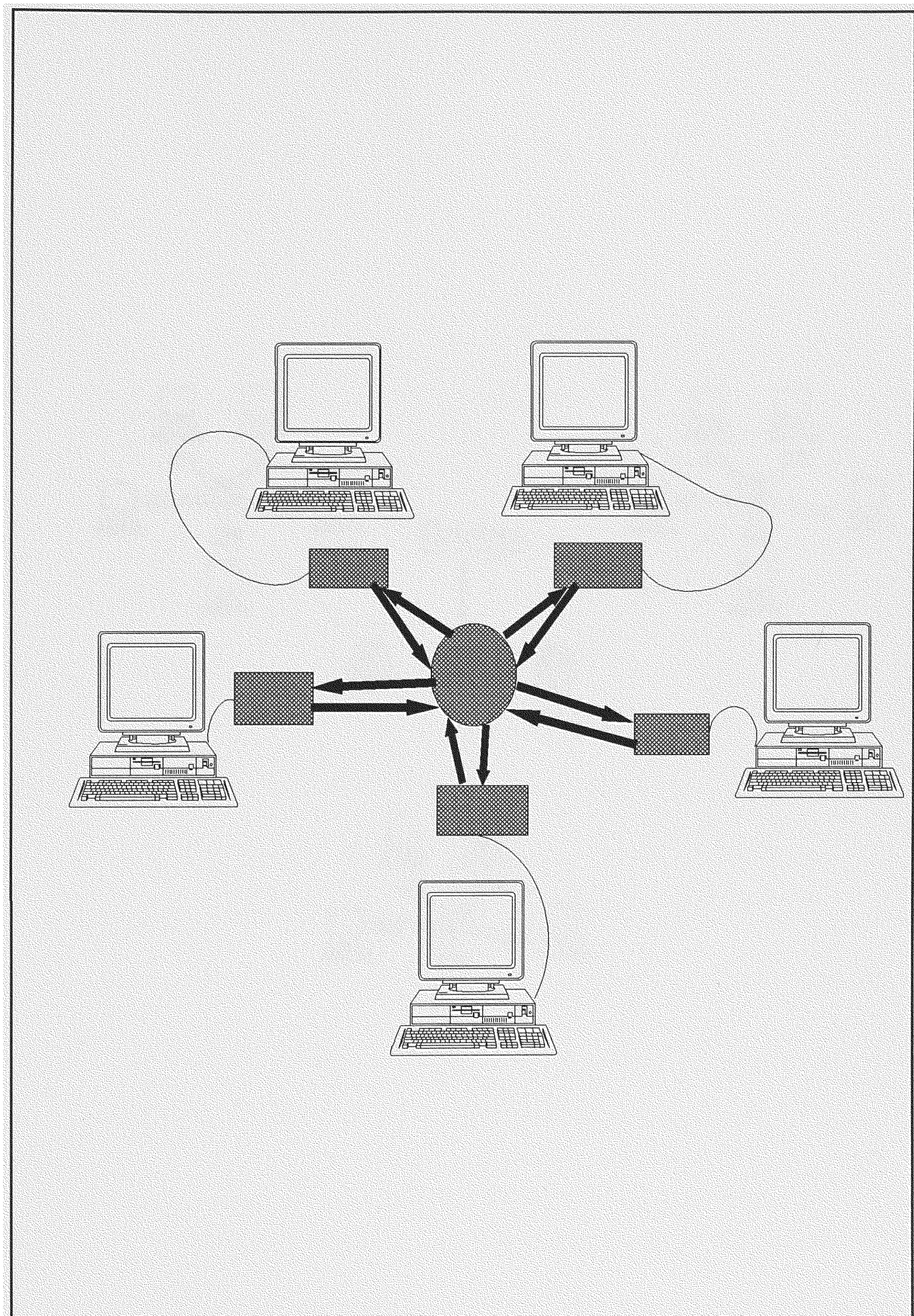

Figure 2.10 Star-Ring Network

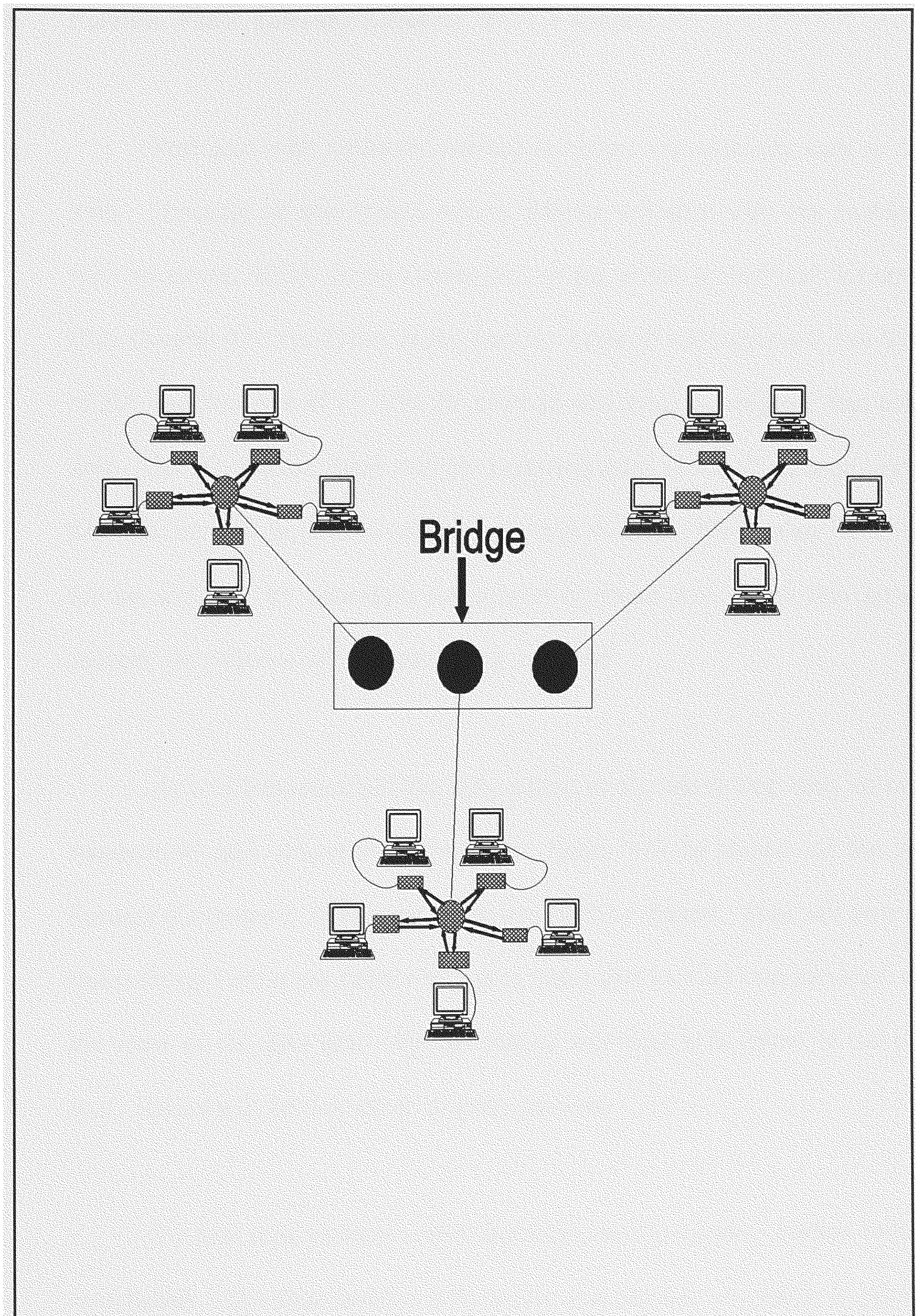

Figure 2.11 Star-Ring Network with Bridge

2.6 Possible Fault Tolerant Schemes in This Project

Problems with errors in received messages are relatively easy to deal with. Choosing an appropriate coding scheme will deal with any problems with bit errors. ISDN frame format will be put to use to deal with bit errors (See Chapter 4). The ISDN frame has two bytes of a B1 channel, two bytes of B2, and four bits of D. The B1 channel will have the original data being sent. The B2 channel bytes will be a negated echo of the B1 channel bytes. The stations will be able to compare the data on B1 to the data on B2 to test for bit errors. This decreases throughput by about half, but it also greatly reduces the chances of bad data being received.

As mentioned earlier, the very nature of the token ring adds an extra measure of fault tolerance. The ISDN frame will be passed all the way through the network and return to the transmitting station. This will give the transmitting station the chance to test to make sure that all data received was the same as the data sent. Special tokens will deal with errors in the data sent. These will be described in later chapters.

To deal with problems with the token, a ring monitor system will be established. The first monitor will be the one defined as station 0. It will send out the start up token on a timed schedule. The time between these tokens will be based on a timer interrupt. The timer will be set to a time based on the number of stations, data rate, number of bits in the frame (48), and a small amount of time added to offset processing time at each station. This process will be used to deal with the possibility that not all stations will come on line simultaneously.

After the initialization token has been passed around, station 0 will no longer be the monitor. The system will be policing itself. Using timer interrupts, the stations will be able to determine if it has been too long since the last byte of data has passed through the network. Warning and error tokens will be used to determine if the failure was caused by a simple fault such as the last transmitting station failing to send a token or a station failing to pass a byte to the next station, or if a more serious hardware fault has occurred. Hardware faults can then be dealt with by skipping the station that has experienced the problem. The process will be described in more detail in later chapters.

To keep a station from sending messages that are too long and are unfairly tying up the network, the transmitting and receiving stations will both count the number of bytes or frames being sent. If the packet becomes too large, the transmitting station will be forced to stop sending. This will be done by using a special token. If this fails to stop a station, or if it is not the station that is transmitting that is sending out the endless message, a second timer interrupt, based on the time between new message tokens, will be set. In a procedure similar to that of a network which has gone quiet, this timer interrupt scheme will track down the station which is generating the data and skip over it. Again, see the software chapter for more details.

2.7 Considerations for Token Rings and Other LANS

Requirements of networks have been steadily increasing over the past several years. By 1995, audio and video will become just another data type. Multimedia is becoming more and more desirable, and networks need to keep up with the increased demand. Multi-media will require much more memory, storage capability, network bandwidth, and processing rates. 100 Mbps will become the baseline for networks, especially those requiring multi-media applications.

Early LAN schemes, such as Token Rings, FDDI, and Ethernet, were not designed to handle the requirements of multimedia. However, voice is not necessarily a constant stream of data. It is actually a mixture of talkspurts and silent periods. This allows for packetizing and sending of the data. Token Rings can take advantage of this fact. There are three current methods for voice-data integration on a Token Ring network.

First, distributed control can be used on the Token Ring. one such method removes the need for centralized control by giving each station a window in which it can transmit. If a station does not receive the token within this window, it is not allowed to send its packet. This allows the synchronous transmission of voice. This window can be either a fixed value, or can be based on the previous transmission.

Secondly, centralized control can be used. In this case, stations can be divided up into two categories: the ones which send only voice, and the ones that send only data. The central node sends out priority tokens at regular intervals. This token is sent around the ring to give all of the voice stations a chance to send a single voice packet. Once every station has had a chance, the central station sends out the regular token. If the regular token is busy, the central station must wait until a free token is sent before it can send the priority token.

Finally, a dual ring can be used. Once again a central station is used. In this case, three tokens are used. One is a null token to warn the stations that a new service cycle is beginning. The second token is used to indicate that only voice stations can send. The third token is used to indicate that only data stations can send.

Compared to other LAN's, Token Rings are preferable when trying to deal with the voice/data integration. For example, carrier-sense multiple access with collision detection (CSMA/CD), another major LAN, has many disadvantages. One of these disadvantages is that making the necessary changes to the CSMA can be very expensive. Another disadvantage is that high utilization of this type of network causes the performance to degrade very quickly. Also, access delay is not guaranteed to be bounded for CSMA, causing probable real time delivery of voice packets and making CSMA not suitable for voice transmission.

Eventually, Asynchronous Transfer Mode, a special switch, will be used to integrate data, voice, and video. However, in the interim engineers have developed other methods of sending video. Video can presently be added to a network using a videoserver. This usually utilizes one or more computers to compress the frames of video signals so that the video can be integrated onto the network. Because it must handle both A/D conversion and data compression, a videoserver must have quite a bit of specialized hardware.

With the need for higher bandwidth on networks has come an increase in the development of viable options to make faster networks. Today's networks are on the verge of broadband (4 Mbps for Token Ring and 16 Mbps for Ethernet). Higher speeds are needed though. Proposals for new networks include 16 Mbps for Token Rings and 100Mbps FDDI (Fiber Distributed Data Networks).

One method of increasing speed on networks is the Broadband ISDN. BISDN can use ATM to perform high speed switching (45Mbps to 155Mbps). The push for this type of speed comes from the need to send large files and animated graphics over large distances. BISDN is also being used for basic consumer use in the form of high definition television (HDTV).

With the advent of ATM, variable length data frames becomes a very important and viable option to increasing network efficiency. The network will no longer have to wait on dummy information that had to be sent to fulfil the synchronous requirements. In older networks, the other stations had no way of determining when a transmitting station was finished, so that packet had to be a definite length. This is a waste of valuable bus time. In this project, variable length packets will be allowed. The maximum will be 256 bytes due to space restrictions on the 68HC11 board, but a pakcket can be any size less than that. The use of tokens will allow the stations to realize when the last frame of a transmission is arriving.

Frame relay, or the sending of variable length units of data through a network, was originally designed as a service for ISDN. Now extensive effort is being put into trying to make it a technical solution viable for any type of service.

Speed is not the only factor in developing technology. Fault tolerance must also be considered. Fault tolerance is not an all or nothing consideration. Each LAN must weigh the options and decide just how much fault tolerance is required. Network fault tolerance is dependent on three key aspects: equipment, design, and installation.

Fault tolerance by design is very complicated. In the design, consideration must be given to how to mix networks, hubs, protocols, and media. The scope of the network and importance of individual parts of the network should be taken into consideration. Splitting up a network into sub networks can help keep the entire network from failing when a single user fails. Parts of the network with more vital functions may require the extra cost and equipment necessary on developing a redundant system; however, this does not mean that every part of the network must use redundancy.

An example of fault tolerance by design is in the choice of network topology. By decreasing the number of hops (connection between networks) between links and workstations, a topology can decrease the chances of something going wrong in the transfer of data.

The network backbone is a prime candidate for redundancy since it is the link which ties subnetworks together. Redundancy can be accomplished by including a second cable which ensures that if one of the cables is broken, the network will still be able to communicate. Also, adding an extra, simple, redundant hub can keep separate hubs and subnetworks communicating when one fails.

Since most errors in a network occur in the hardware, equipment is also an important choice when considering fault tolerance in a network. Power supplies are appropriate places for adding redundancy to network components. There must also be equipment included which will allow switching of control when a fault occurs.

Of course, fault tolerance on the network level is useless without fault tolerance on the station level. More modern techniques can target specific areas or devices for fault tolerance. For example, faults in control computers can be monitored using band limiting filters. These filters are employed to monitor the propagation of a signal through the device. By comparing the signal to a signal from a another source, the system can detect if the signal has exceeded a threshold. This would indicate a possible fault in the system.

Chapter 3

System Hardware

3.0 Introduction

In this chapter, the key element of this project, the chosen hardware, will be discussed. The first section of the chapter will cover the architecture of the 68HC11 EVB board. After that will be a discussion of the RS-232C and its hardware. The next section will discuss the PC's which will be used as terminals on the network. The final section will be a discussion of using microcontrollers to make network gateways.

3.1 The 68HC11 Microcontroller Unit

3.1.0 Introduction

In choosing the hardware for the repeaters, there are several considerations. The first consideration is the intelligence required. The most important function of these repeaters will, of course, be the sending and receiving of messages. This means dealing with source and destination addresses, changing the token between free token and busy token, dealing with priority schemes (if any), etc.. The repeaters for this project must deal with the possibility of being a monitor, which means that they must be given timing schemes and instructions on skipping the next station if a fault is detected. They will also probably be used for coding and parity generation. For all of these reasons, the repeaters need to be reasonably intelligent.

The next factor is cost, reliability, and availability. The repeater must be made with accessible materials to make the network worthwhile. Designing with excessively high priced or hard to find materials would make building the network next to impossible for almost everyone who wishes to build it.

The final factors in choosing the hardware for the repeaters are convenience and flexibility. The hardware chosen should be relatively well known and user friendly. This helps with maintaining and managing the network. The more familiar a user is with the hardware the easier it will be to fix small problems with the network or to make any necessary changes to the network to make it fit the user's needs. The user will not want to have to call in an expert for every problem that occurs.

The Motorola 68HC11 is well suited to implement this network. Introduced in the late 1970's, it has become one of the leading and standard microcontrollers in the industry. The 68HC11 is intelligent enough to deal with all of the functions that will be required of the network's repeaters. The microcontroller has 108 instructions, plus several versions of many of the instructions. This makes it very flexible and easy to program. Furthermore, the 68HC11 is easily obtained and is relatively inexpensive. As of the Spring of 1993, the 68HC11 sells for just over $130 for a single unit. When bought in quantity, the cost can drop to $79 per unit. Since it has been around for almost two decades, the 68HC11 has been tested and updated enough to remove most of the major bugs. The 68HC11 is a common microcontroller, so many software and hardware Engineers have had at least some experience with it. Even without any experience, the 68HC11 is user friendly enough that anyone who has had any experience with microprocessor boards should not have much trouble in learning the system. Again, its age benefits by having reference material for the 68HC11 not only readily available, but also accurate and quite thorough.

3.1.1 General Information

The MC68HC11A8, the 68HC11 being used, is an 8-bit microcontroller with the ability to simulate some 16-bit functions (see the next section for details). The microcontroller is capable of average bus speeds of 2 Megahertz; however, it is also capable of bus speeds down to DC levels. This fact, combined with its HCMOS (high-density complementary metal-oxide semiconductor) VLSI design, means that the 68HC11 consumes little power compared to other microcontrollers. For on board memory, the MC68HC11A8 has 8 kilobytes of ROM (read-only memory), 256 bytes of RAM (random-access memory), and 512 bytes of EEPROM (electrically erasable programmable ROM). The board itself has an additional sixty-four kilobytes of address space, which can be used for storing programs, stacks, queues, etc.

The 68HC11 comes with some of its own on-board fault tolerance. There is an illegal operation code (opcode) circuit designed to detect any illegal opcodes and to cause a nonmaskable interrupt in case one is found. In case something happens to the clock, a monitor system is available to reset the system. Protection from software failures is provided by a watchdog system.

The diagram on the next page shows the general layout of the token ring designed for this experiment. Each station will consist of two HC11's. The first will be connected to the user's computer and will also act as an input to the network from the station. This will be referred to as the inner HC11. The second HC11 has no direct contact with the computer and serves as the input to the station from the network. This will be referred to as the outer HC11.

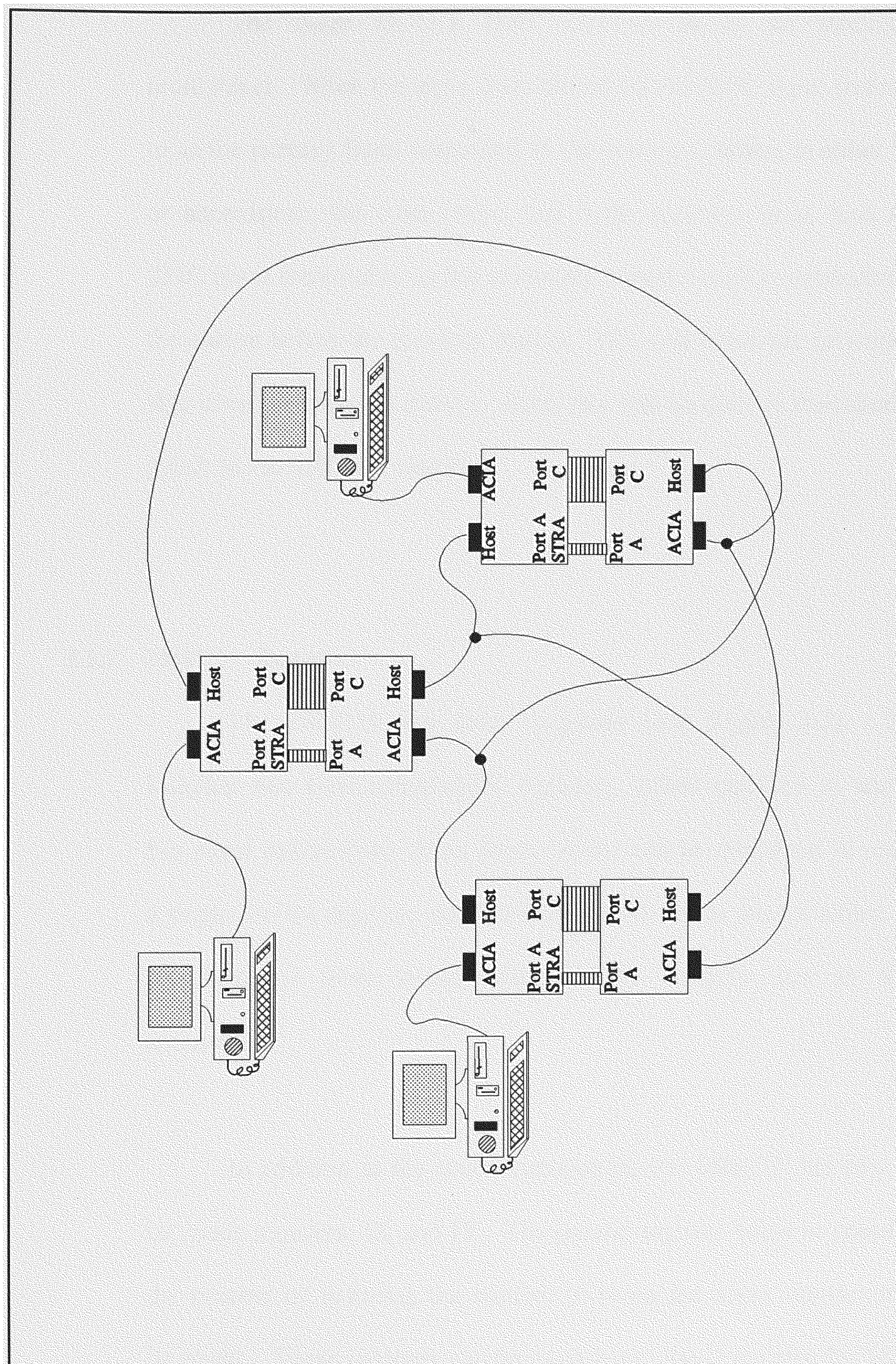

Figure 3.1 Network Design for this Experiment

The outer HC11's basic function is as an intelligent multiplexer. When the input from the ACIA RS-232C input (referred to as the primary line), connected to the previous station, is either bad or nonexistent, the outer HC11 will begin listening to its host RS-232C input (referred to as the secondary input), which is connected to the station before the previous station. This will cause the network to skip over the previous station. How this will be done is described in detail in the software chapter.

3.1.2 68HC11 Registers

The 68HC11 has seven registers (see Figure 3.2). This includes two 8-bit accumulator registers, called registers A and B. For many instructions, these accumulators can be combined to create a single 16- bit register, called the D register. This allows added 16-bit functions even though the 68HC11 actually has an 8-bit architecture.

In addition to the two accumulators, the 68HC11 has two 16-bit index registers, IX and IY, The second register helps to speed up the process by reducing the number of times the index register must be saved. These registers can be moved into the D register for 16-bit arithmetic operations.

The final three registers are a 16-bit stack pointer, a 16-bit program counter, and a standard 8-bit condition code register. The bits in the condition code register are defined as follows: zero (Z), negative (N), stop disable (SD), X interrupt mask (XI), I interrupt mask (IM), carry (C) and half carry (HC), and overflow bits (O).

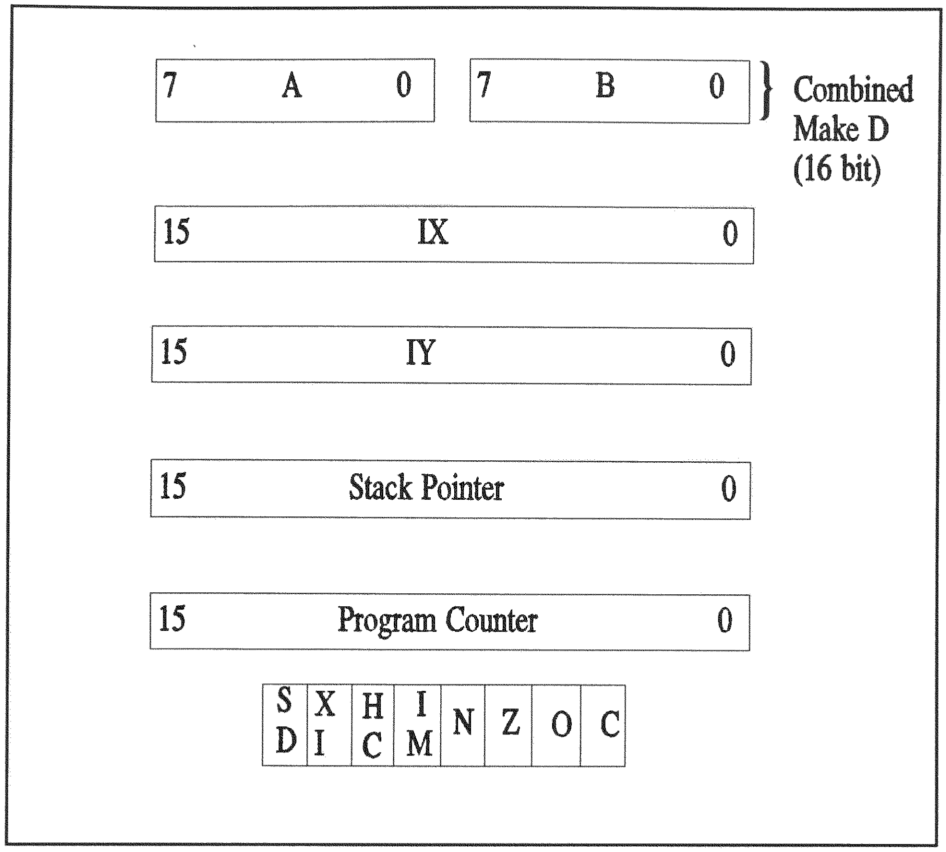

Figure 3.2 Registers of the 68HC11

Table 3.1: Summary of HC11 Register Functions	
Register Name	**Function**
A, B	Two 8-bit accumulators. Combine to make one 16-bit accumulator (D).
IX, IY	Two 16-bit index registers.
Stack Pointer	A 16-bit register which points to the current working stack.
Program Counter	A 16-bit register which points to the next line of a program to be executed.
Status Register	8-bit register which holds important execution data.

3.1.3 68HC11 Ports

The HC11 EVB board has a total of 7 ports. Of these, four factor into this thesis: Port A, Port C, and the two RS-232C ports (ACIA and Host). These four ports will be discussed in this section. All registers to utilize the ports are memory mapped. This means that the programmer must manipulate certain addresses on the EVB board in order to use the ports. See Figure 3.3 on the following page for connections between the boards.

The ACIA is one of two asynchronous serial I/O ports. It is utilized with software by manipulating two registers: The ACIA status and data registers. If a byte of data has been input into the ACIA port, the first bit (LSB) of the ACIA status register will be changed to a logic 1. The data is read into the microprocessor by reading the ACIA data register. The first bit of the ACIA status register will remain a 1 until the status register and then the data register have been read. These registers are located at address locations 9800 and 9801 (hex), respectively. To output from this register, the second bit must be tested. If it is a zero, the port is not ready to output. If the second bit is a logic 1, the user can write to the data register and the data will be sent. For the inner HC11, this port will be used to communicate with the host computer. On the Outer board, the ACIA will act as the primary input from the network.

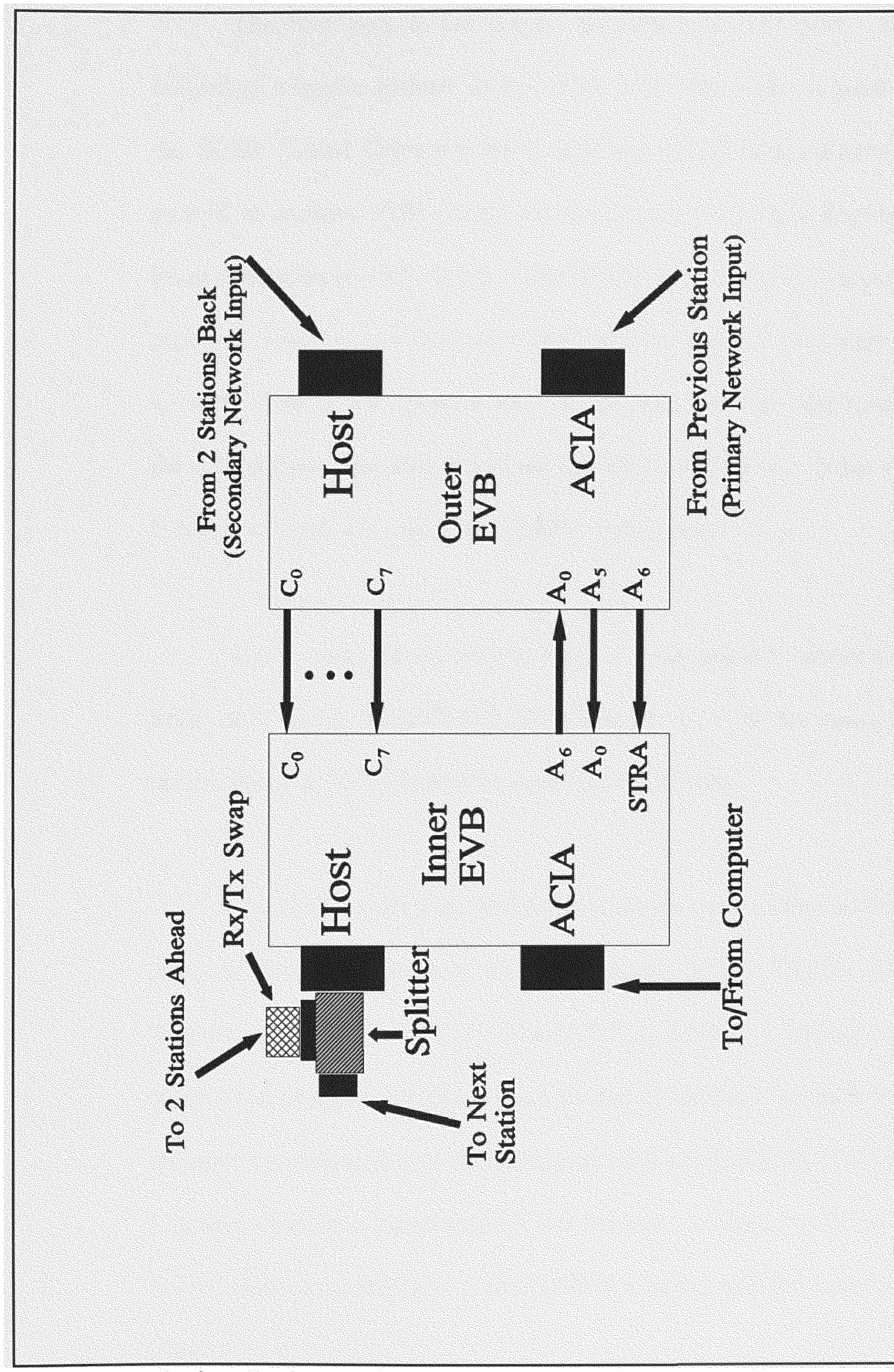

Figure 3.3 Station Hardware Design

The host port is the second asynchronous I/O port. It is utilized in a similar manner as the ACIA port. The registers used are the SCSR (Serial Communication Interface (SCI) Status Register), located at address 102E (hex), and the SCDR (SCI Data Register), located at address 102F (hex). The SCDR is actually two separate registers. One is used only on a read, and the other is used only on a write. If there has been an input to the host since the last read of the data register, the sixth bit (from the LSB) will be set. If the port is ready to send, the eighth bit (MSB) will be set.

The C and A ports will be used to communicate between the inner and outer MC68HC11EVB boards. A will be used for handshaking, C will be used to send the actual data.

The C port is an 8-bit parallel port which can function as either input or output, depending on the contents of the DDRC (Data Direction Register for port C). The DDRC is located at address 1007 (hex). The programmer can make a certain bit of the C register to be an input by writing a 0 to the appropriate bit of the DDRC. A logic 1 is used to make a bit an output. For example, writing hex 0F to the DDRC will make the first 4 bits (LSB's) outputs and the other four inputs.

The C port has two data registers: The PORTC and PORTCL. The PORTCL data register actually latches the data after a clock edge (negative or positive edge can be chosen by the programmer) is applied to the STRA pin. The PORTC register changes along with the port C pins. PORTC is located at the address 1003 (hex), PORTCL at 1005.

The PIOC register is the control register for the C port. To determine if a byte has been latched into PORTCL since the last read, the programmer can test the most significant bit of this register. If it is a logic 1, fresh data is in the latch. Otherwise, there are no new data.

Data will be transferred between the HC11's using the PORTCL register. The inner HC11 will have its C register set to input, the outer HC11 will use the C register as output.

Finally, the A register is half input, half output. The first three pins are input pins, the next four are output pins, and the last pin is an input or output, depending on the data direction register for A. Two output and one input pin will be used for the outer EVB. One input and one output will be used by the inner EVB. Reading or writing to these ports is accomplished through the data register for port A, located at address 1000 (hex).

3.1.4 68HC11 Failure Statistics

Motorola supplies failure rate statistics for its microcontrollers and microprocessors in the Motorola Microprocessor, Microcontroller, and Peripheral Data data book. In this section, a very brief summary of these results will be supplied. See the bibliography for information on the data book.

Motorola runs several tests on its microprocessors for this report. One of these is the data retention test on the EPROM and EEPROM. This is a test of the EPROM and EEPROM's abilities to hold a charge over an extended period of time. The 68HC11A8 showed a failure rate of about 0.34% for up to 1008 hours at 150°C.

Another test is the thermal shock test, in which the chip is tested by moving it directly from a fluorocarbon bath at -65°C to a bath of 150°C. This test the stress caused by sudden changes in temperature as well as the increased conductivity due to a liquid environment. This resulted in a failure rate of 0.13% for the HCMOS family of chips.

The third test is the high temperature operating life test, which is performed to accelerate the failures due to the application of extreme conditions, including high input voltage and temperature. Tests ran for a maximum of 1008 hours with a voltage of 5.5 volts and temperature of 125°C. These resulted in a failure rate of about 0.036% for the MC68HC11.

The EEPROM read/write cycling test measures EEPROM cell operation over an expected lifetime. After running the 68HC11 at 5.5 volts and 85°C, a failure rate of 0.87% was found.

There are several other test results available in the data book, but they all point to the same thing: The 68HC11 has a small failure rate even under extreme conditions.

3.2 The RS-232C

3.2.0 Basic Information

Obviously a vital part of any network is the means by which the devices on the network will be connected together. For inter-connection between stations on the network designed for this project, the RS-232C was chosen because it is the most common interface standard. In signaling, the RS-232C uses two voltage levels: A twelve (12) volt level is used to represent a logic 0, and a negative twelve (-12) volt is used to represent logic 1. In reality, voltage is lost as a signal passes through the connection line. Therefore, anything above three (3) volts is considered a logic 0, and anything below (more negative than) negative three (-3) volts is considered logic 1.

The RS-232C lines are generally good for signals up to 20Kbps. Furthermore, distances traversed by a single RS-232C line should not be greater than 15 meters. Voltage levels begin to drop too far and signals begin to get garbled for signals faster than 20Kbps or for distances greater than 15 meters. Distances can be made greater by boosting the signal at points along the line. Faster data rates can be achieved with good design, including data compression techniques. The data rate limit is the reason for the 16Kbps data rate limit of the HC11.

The basic RS-232C connector has 25 pins; however, most applications do not require all of these pins. The HC11 RS-232C ports use only a few of these pins for communications. Figure 3.4 illustrates the pins used by the HC11's two RS-232C ports. Table 3.1 briefly describes the use of each of these pins.

Table 3.2: RS-232C Lines Used by the HC11EVB	
Name:	**Function:**
GND - Protective Ground	Attached to machine frame or some external ground
RXD - Received Data	Serial Data Output Line
TXD - Transmitted Data	Serial Data Input Line
CTS - Clear To Send	Output Signal Used to Indicate that the Device is Ready to Transfer Data.
DSR - Data Set Ready	An Output Line Used to Indicate that the Device is Ready to Transfer Data
SIG-GND - Signal Ground	A Line Used to Provide a Common Ground Between Devices
DCD - Data Carrier Detect	An Output Line Used to Indicate that an Input Signal Has Been Detected
DTR - Data Terminal Ready	An Input Line Used to Indicate On-Line/In-Service/Active Status

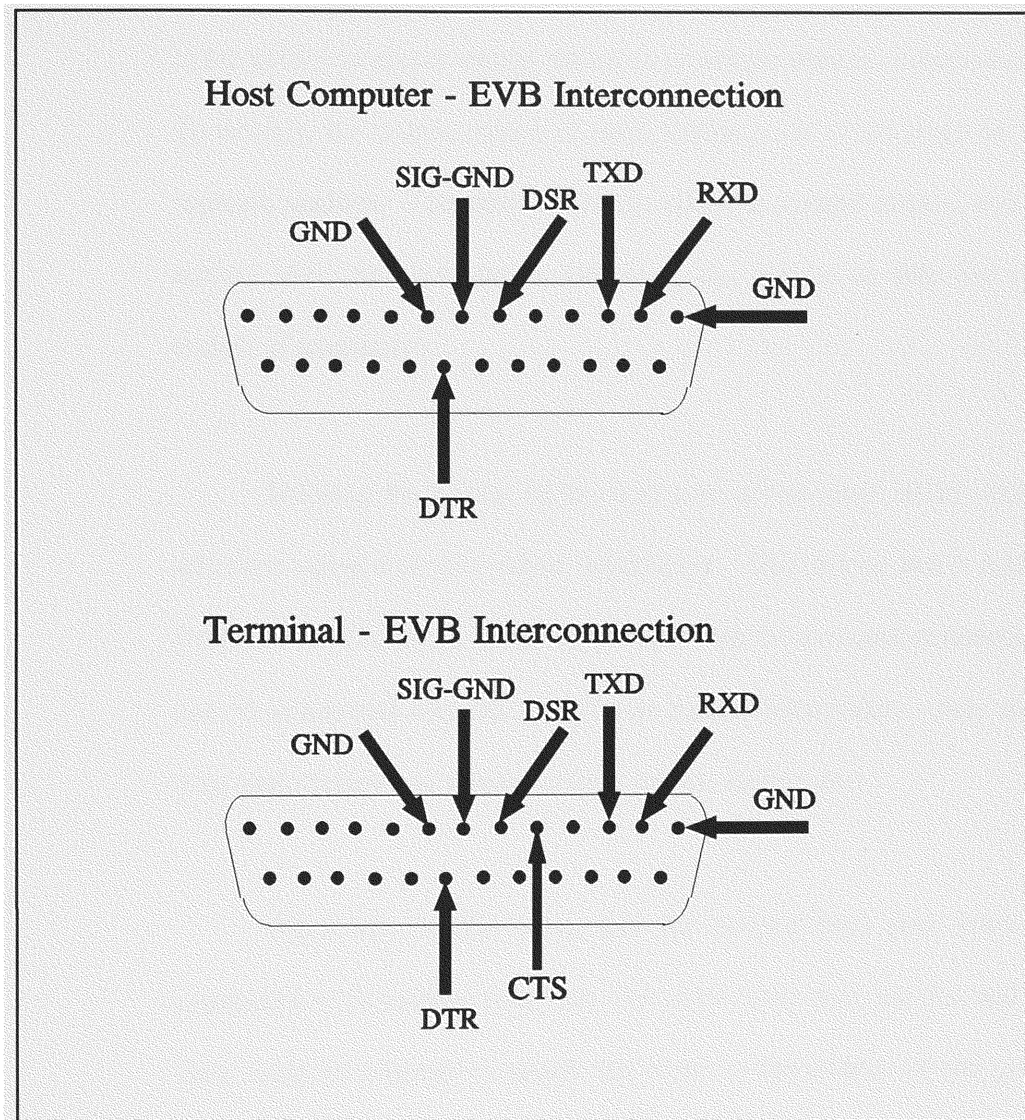

Figure 3.4 RS-232 Pin Layout for the HC11

3.2.1 Splitting the RS-232C

The most important fault tolerant characteristic of this design is the ability to skip a station which is no longer able to communicate. To do this, the output signal of each station's HC11 must go to two separate stations: The station directly after the present station, and the station after the next station (allowing the system to skip the next station if necessary).

Splitting the output of the host port of the inner HC11 will be achieved using a device called Multi-port. This device has 1 male/3 female connections, all wired in parallel. Up to two input connections can be connected directly to any one output connection. Any more than that creates too much loading on the output port.

Also, one of the ports that the output host port must be connected to is another host port. In order to do this, the Rx and Tx lines must be crossed to match the input to the output of the ports. This will be done using a device called a reverser.

3.3 IBM PC As a Workstation

As stated earlier, IBM Personal Computers will be used as the workstations on the network. Their main function will be to send data and programs to the 68HC11's, which will be the repeaters of the system.

Programs for the 68HC11 (network, coding, etc.) were written using some type of word processing program. The programs were assembled on an assembler designed for the IBM PC by Motorola. Using a communications program, the assembled programs were then downloaded into the 68HC11. The final network function of the IBM workstation will be to start the network program on the 68HC11 running. Specific programs that will be used for communications and word processing will be discussed in the next chapter.

After this was done, the computers were used purely for data input/output. The PC's are used as dumb terminals; data are not sent into their memories; however, the data are echoed to the screen so the user can see what is being typed. Data are downloaded to the 68HC11's through the same communications program mentioned above. Data are sent directly from the Personal Computers' keyboards to the 68HC11 through one of the serial (COM) ports supplied on the PC's. Data received from other terminals are dealt with by the 68HC11's, then echoed on to the personal computers' screens.

3.4 Network Gateways By Microcontrollers

As stated earlier, the 68HC11's act as repeaters, which serve as the network gateways for the stations. This is accomplished by done using the two RS-232C compatible serial ports on the 68HC11 board. These were chosen instead of the other ports (there are five other ports on the 68HC11) because RS-232C is commonly used in computer communications.

One of these ports is used to connect the 68HC11 to the IBM Personal Computers. The communications is set at 9.6 Kilobits per second. Communications software on the IBM PC is utilized to establish and hold the link. Information on the communications software, exactly what the 68HC11 does with the IBM PC, etc. will be discussed in the next chapter.

The second serial port is used for communications between the 68HC11's. It can be seen from the token ring schematic in Chapter 2 that the repeater has to be connected to the network in two directions: One direction for data transmission, the other direction for data reception. For communications on the network, however, there is only one RS-232C port available. Therefore, this one port has to be divided into two. A discussion of how this is done can be found in section 3.2.1 Splitting the RS-232C. The data are sent through the network at the safest rate for RS-232C communications on the 68HC11, i.e. 9.6 Kilobits per second.

For the most part, the rest of the functions of the 68HC11 will be accomplished using software. These will be discussed in Chapter 5, which is the software chapter.

Chapter 4

Communication Protocols

4.0 Introduction

There are certain requirements which are vital to the development of any network. One of these is the need for some type of protocol to keep the system from dissolving into anarchy. Another requirement is some type of fault tolerance scheme to keep the network up and running. After all, if the data received on the network are suspect, the network itself is useless. These requirements can be met using hardware, software, or combinations of both. This chapter will discuss these vital elements and how they have been met in this experiment. The discussion will include both references to methods used in other systems and the methods used in this experiment.

4.1 Other Communications Protocols

4.1.0 Introduction

In order for a network to be successful, all stations on the network must adhere to a common set of rules and standards. These protocols define everything from data format to the number of bytes which can be sent during any transmission. This section will discuss typical protocols used in token rings.

4.1.1 IEEE 802.5 Token Ring

One protocol designed for a token ring network is the IEEE 802.5 token ring. Special features of this protocol include a variable length frame, bits set aside for priority, and it allows reservation of the token by a station which wants to send a token.

Figure 4.1 shows the basic frame format of the IEEE 802.5 token ring. The token bit is a 0 if it is a token and a 1 if it is a free frame. The monitor bit is used to keep a frame from continuously circulating around the ring. The priority bits show the priority of the station which is taking the token. Only a station of higher priority may take the token. The reservation bits are for a station with higher priority to set aside the next free token. Once the previous message has been sent, the reservation bits are moved into the priority bit location so that only stations of higher or equal priority can take the token. The rest of the bits are self-explanatory. The IEEE 802.5 protocol delays transmission of the free token until the transmitting station has received the header of message. The token consists of a sequence of starting delimiter, access-control field, and ending delimiter. The information field begins with a header.

SD	PPP	T	M	RRR	FC	DA	SA	Info	FCS	ED	FS

Figure 4.1 IEEE 802.5 Frame Format

Table 4.1: Bits Used in IEEE 802.5 Frame

Term	Definition
SD	Starting Delimiter
PPP	Priority Bits
T	Token
M	Mode
RRR	Reservation Bits
FC	Frame Control
DA	Destination Address
SA	Source Address
Info	Variable Length Information Field
FCS	Frame Check Sequence
ED	Ending Delimiter
FS	Frame Status

The strength of this system is that a priority scheme can be designed. However, it also lends itself to the indefinite postponement of a message for those who are not fortunate enough to have a high priority. Limits should also be set on the information area of the frame to keep one station from keeping the line tied up for too long.

4.1.2 FDDI

A second protocol is the Fiber Distributed Data Interface, designed for a High-Speed Local Network (HSLN). This protocol is designed to take advantage of the high speeds that fiber networks can attain (100 MHZ).

For this protocol, limits on the data sent are determined by timing rather than size in bits. An agreed upon time limit is placed on sending. If a message goes beyond this time, the message is not cut-off. It is simply taken away from the station until the time is made up. Thus, the message length time averages out to be the time limit set.

Below is the frame format. Notice that it is similar to the frame format of the IEEE 802.5. There are no access control bits, however. The preamble is a sequence of idle symbols used for clocking purposes by the receiving stations.

Preamble	SD	FC	DA	SA	Info	FCS	ED	FS

Figure 4.2 FDDI Frame Format

Table 4.2: Bits Used in FDDI Frame	
Term	**Definition**
Preamble	Idle Characters for Timing
SD	Starting Delimiter
FC	Frame Control
DA	Destination Address
SA	Source Address
Info	Variable Length Information Field
FCS	Frame Check Sequence
ED	Ending Delimiter
FS	Frame Status

4.2 Protocols Used in This Work

4.2.1 The ISDN Protocols

This experiment is designed to work on the ISDN frame format. This section will be a description of ISDN protocol.

The primary concept behind ISDN is allowing several channels on one line. Basic Rate Interface (BRI) consists of two "B" channels and one "D" channel. The B channels (referred to as B1 and B2) each run at 64 Kbps and are used for data, voice, etc. transfer. The

D channel runs at 16Kbps and is usually used for control information but can also be used to send data. This gives a basic rate of 144kbps. Another 48Kbps is required to compensate for overhead bits, resulting in a final data rate of 192Kbps.

The basic frame format is shown below. The frame consists of two bytes of B1 data, two bytes of B2 data, four bits of D, and twelve overhead bits. Most of the overhead bits are for voltage level testing which is not necessary for this experiment. However, the bits were left in the frame to simulate the ISDN protocol as much as possible.

F and F_a are the framing and auxiliary framing bits. They are described as being positive zero. Without a negative zero, this does not really make sense. Since the RS-232C uses +12 volts for a logic 0, this bit was set to be a logic 0.

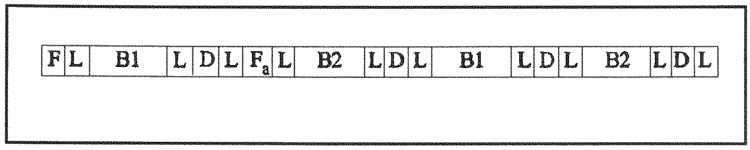

Figure 4.3 ISDN Frame Format

Table 4.3: Bits Used in ISDN Frame	
Term	**Definition**
D	Single bit of D-Channel Data
B1	Byte of B1-Channel Data
B2	Byte of B2-Channel Data
L	DC Biasing Bit
F	Framing Bit
F_a	Auxiliary Framing Bit

The L bits are DC balancing bits and should be the opposite voltage of the F bits, which means that they are generally negative zeros.

The B1 and B2 channels can be used to send data (packet or circuit switched) to separate destinations. For example, one line may be connected to a fax and the other to a phone. In this case, the user can be receiving a fax from one location while speaking on the phone to another location.

For this work, the B1 channel will be the primary data link. The B2 will be used for sending a negated echo of the data on the B1 channel. In the frame format, the data on the B1 channel will be echoed on the B2 channel immediately following it.

The D channel will be used to carry the token. This gives four bits for tokens. This provides for a maximum of sixteen different tokens. These will be listed in the token ring protocol section.

Limits of the HC11 keep this network from running at the 192Kbps. The network will be designed to run at 9.6Kbps, which breaks down to 800hz for the D, 3.2Kbps for each B, and 2.4Kbps overhead maximum data rate. This slow rate is one of the weaknesses of this model, but the data rate can be increased by using a different microcontroller board (see recommendations in chapter 7).

Another weakness of using the ISDN frame format is that the large amount of overhead further reduces the rate of the channels. This is especially bad since these bits are unnecessary for a non-ISDN system. However, this system could eventually be converted to communicate directly with another ISDN system.

The strength of the frame format is that each data frame sent has its own control information. With the other protocols, the data are sent in one big clump. If there is an error in the data, it can propagate throughout the message and could lead to the entire message needing to be re-sent. For this system, there is no risk of this happening. The error can be spotted and re-sent before a pileup of data occurs. Also, since there are two channels, the system could eventually be developed to send two separate messages between two different pairs of stations simultaneously (see recommendations in chapter 7).

4.2.2 Fault Tolerant, Token Ring, and Other Protocols

This network follows the traditional rules of the token ring network. No station can send a message until it receives a free token, thus keeping more than one station from sending at a time. Each station inputs from one direction and outputs in the other. The entire network is a closed system and cannot be broken into once it is running.

Some of the features added to this system include a special "boot up" token which provides several special functions. When a station receives the token for the first time, it takes the number in the first byte of B1 in the frame and increments it. It then stores this value as its address. Since the data travels through both HC11's in each station, both HC11's can find out their address in this without having to directly communicate. Next, each station knows the number of the previous station and the next station. This helps later on in the fault tolerance software. The start-up station (station 0) sends the token around a second time. Since the token had the stations counting off, this second trip around will provide the stations with the total number of stations on the network, which can help with the timer calculations (see software chapter) and can be used for security purposes.

Another feature incorporated is the use of several different tokens (which actually act as control bits). This helps with software and fault tolerance. The following table lists all of the possible tokens and a brief description of what each does. These tokens will be described in greater detail in the software chapter.

Messages will be limited in two ways. First, the packet length for the messages will be 256 bytes, or 128 frames. This will be maintained by counters in both the transmitter and the receiver. However, retransmissions will not be counted in the byte/frame counters. Therefore, to keep a station from continuously sending and/or retransmitting, the stations will also have a certain time limit. Transmitting stations will be cut off if they go beyond either the byte or time limits. To see how this will be done, see the software chapter.

Table 4.4: Tokens and Their Functions	
Token (binary)	**Meaning of Tokens**
0000	Free Token
0001	Busy Token, Source in first byte B1, destination in second byte B1
0010	Warning token ... Something maybe wrong with the network (someone sending too much, or network has gone silent). Address of Discovering Station on first byte B1.
0011	Warning: Sent too many bytes of data. Stop Transmitting.
0100	Received busy token acknowledgement.
0101	Error token. Cut station out of network (Station address + 1 in byte B1)
0110	Initialization Token. Has address and number of stations data on B1.
0111	Busy Token: Broadcasting to whole network.
1000	Received good data acknowledgement.
1001	Data being sent to Receiver.
1010	Retransmitted data after request.
1011	Last byte of transmission.
1100	Retransmit request from receiver.
1101	Retransmission from transmitter. Error in returned data. Frame number in B1.
1110	Last 2 bytes of transmission.
1111	Retransmission received.

Chapter 5

System Software

5.0 Introduction

Chapter 5 presents a study of the software modules necessary for this thesis. Topics will include communications between a station's inner and outer HC11's, between the inner HC11 of one station to the outer HC11 of the next (the actual network) and PC to HC11 communications.

5.1 Token Ring Inner HC11 Software

5.1.0 Introduction

Since they act as the repeaters in the token ring, the heaviest part of the communications is dealt with by the inner HC11's. For this reason, the most complicated and longest programs are written for the 68HC11 microcontrollers. Since the Random Access Memory (RAM) space in these units is limited, the routines were written in assembly code (writing directly into assembly code makes for more efficient code than writing in a higher level language and compiling). In this section, the modules for each function of the inner HC11's will be described in detail.

5.1.1 Inner HC11 Main Program

The inner HC11's, which are the microcontrollers between the network HC11's and the computer, have the longest and most complicated programs of the system. The HC11's are programmed to loop in a main routine which polls the ACIA for input, polls the C port for data from the network, tests to see if something needs to be sent to the screen, and polls to see if something needs to be sent to the network. When any of these functions are applicable, the HC11 program then proceeds to an appropriate sub-routine. These functions will be described in detail in the following sections.

5.1.2 Input From Terminal

Once the network programs start up, the personal computers used in this thesis basically act as dumb terminals, doing very little on their own. Therefore, one of the main functions of the HC11's is to gather the data from the terminal and store it until it can be sent on to the network.

This part of the program uses polling of the ACIA to input from the keyboard. When the terminal serial port (ACIA) receives data in its buffer, it sets the first bit of its status register. The character is then read from the data register of the ACIA. If it is the first character taken since the last queue was sent using the "control z," or not sent using the "control x," character (see below), the character is considered to be an integer and is used as the address of the destination of a message. Otherwise, depending on the data, the HC11 then performs one of several possible functions: Put the data into one of the two output queues after echoing it to the screen, ready the data for sending on to the network and wait for a free token before sending it, or "lose" the data (if the message has been aborted).

If the input is a character, the byte is added to a queue. As stated earlier, there are two output queues. The HC11 alternates between the two queues for storage. Since the HC11 must have the free token before it can send a message, there may be a short wait before the message can be sent. The two queues allow the user to begin a second message while the HC11 is waiting to send the first. Queues were chosen (as opposed to a queue) because of their First In, First Out (FIFO) system: The first character placed into the queue is the first character sent onto the network.

If the character is a "control z" (control button and z pressed simultaneously) the HC11 goes into Output to the Network mode. This is discussed in more detail in a later section. Meanwhile, the active queue moves to the second queue to deal with any further input from the terminal.

The third possibility is that the input is a "control x", which is the abort message signal. The HC11 places the head and tail of the queue to the same address, thus erasing the queue, and resets all functions so that it will be ready for the next message.

The final possibility is the back space character. If this character is typed, the last character placed in the queue is removed by moving the tail of the queue up one address. This character, of course, is not saved in the queue.

The inner HC11 is also used to set up the station's screen. The screen is split in half: The top half displays data from the station going to the network, the bottom half of the screen displays data coming in from the network. The screen setup is accomplished using the ANSI codes to place the cursor.

5.1.3 Output to Terminal

The HC11 is also responsible for the output of a received message to the screen. When a message is received, it is placed into a special queue set aside for network input (more on this queue later). As long as the data are valid, the HC11 begins removing the characters from the queue and sending them to the terminal.

Most of the network input is dealt with in the outer HC11. This HC11 uses the C port to pass the contents of the D and B1 channels to the inner HC11, which then proceeds to perform the function specified by the D control nibble. If the station is receiving and the data are valid, the D nibble may represent a 9, 11, 14, or 15. This indicates that the HC11 should add the 2 bytes of the B1 channel to the queue. In the case of the B and E control nibbles, the HC11 is programmed to set a special control variable which places the program into Output to Terminal mode, which prints the contents of the queue to the screen for the user to read.

Certain error messages (see fault tolerance section on the inner HC11 for details on these errors) can also be sent to the screen using the Output to Terminal mode of operation.

The ACIA output also uses polling. When the ACIA is ready to output the next byte, it sets the second bit of its status register. When this bit is set, the HC11 sends the next character to the queue to the ACIA data register, then move back into the main program. If the bit is not set, the HC11 returns to the main routine to perform any other functions.

5.1.4 Output to Network

Data on the network is sent from the transmitting station's inner HC11's host port. When this port is ready to send something, the most significant bit of the SCSR status register is set to a 1. Therefore, the HC11 is programmed to first test to see if something needs to be sent onto the network using a special variable set aside for that purpose. Then, if there is something to be sent, test the first byte of the SCSR to see if the host port is ready to send something. If the bit is not set, the program waits for it to be set before sending. If they are set, the data are placed in the host data register, the SCDR and then the program moves back to the main routine.

If the station has a message to send and has received the free token (0 on the D nibble), it proceeds to Output to Network mode. First, it sends a frame consisting of a 1 on the D, the source address, and the destination address. After receiving the same data back as a positive acknowledgement (D nibble of 4) it begins sending it the data. The data are placed into ISDN frame format and then sent a byte at a time. As stated in the previous chapter, the ISDN frame is six bytes long and each frame carries two bytes of data. The entire message is be sent out in this way. Once the output queue is empty and all of the data have been tested (see fault tolerant protocols later in this chapter), the HC11 builds and sends a free token frame.

If the HC11 belongs to the receiving station, it converts the recently received data into ISDN frame format using a D nibble of 8. Then the station moves temporarily into Output to Terminal mode and send the frame back to the sender.

If the HC11 belongs to neither the transmitter nor the receiver, it receives the data from the network as an ISDN frame, become a transmitter temporarily, and sends the data on to the next station.

5.1.5 Receiving Data from the Network

The inner HC11 receives data from the network through the C port. The C port is a general purpose parallel input/output 8-bit port. For the inner HC11, it is set to input only using the data direction register, DDRC. Port A, with pins A_2 - A_0 inputs, pins A_6 - A_3 output, and pin A_7 input or output, is used for handshaking between the two HC11's.

By placing a logic 1 on pin A_6, the inner HC11 indicates to the outer HC11 that it is ready to receive from the C port. This pin is connected to the A_0 pin of the outer HC11. As long as this pin is 0, the outer HC11 does not send to the inner HC11.

Pin A_0 of the HC11 is connected to pin A_5 of the outer HC11.

This pin is used to indicate if the byte sent is D and B1 data (a logic 0) or ISDN frame format (logic 1). If the data do not affect the station, the outer HC11 sends the data directly through to the inner HC11 as an ISDN frame so that the inner HC11 does not have to waste time converting the B and D data back into frame format.

Finally, the STRA pin of the inner HC11 is connected to the A_6 pin of the outer HC11. When this pin receives a negative clock, the data on the C port is latched into the port C latch register (PORTCL).

After receiving the data and determining if it is ISDN frame format or D and B1 data, the inner HC11 performs the appropriate functions on the input. In the case of ISDN frame format, the HC11 reads the six bytes in, puts them into a queue, then goes to Output to Network mode to send them out. If it is not ISDN mode, the HC11 is programmed to test the first byte (which is the D nibble) sent over to determine the function it must perform. There are three queues for the input from the C port: One which holds the last complete message (waiting to be sent to the screen), one to hold the latest incoming message, and one small queue to hold a 6 byte ISDN frame which is passing through the inner HC11.

A summarized description of the tokens can be found in Table 5.1 on the following page. The following is a detailed description of each token and the action taken when each token is received.

If the D is a 0 (free token), the HC11 determines if there is a message to be sent. If there is, the HC11 puts itself into Output to Network mode, sends out a busy token frame with the source and destination addresses, and begins sending its message. If there is no message to be sent, it sends the free token out.

If the D is a 1 (busy token), the HC11 will prepare its input queue to receive data from the network, including pushing the address of the source into the first byte of the queue. Notice that the outer HC11 does not send this busy token to the HC11 unless it is for this station's address. The inner HC11 does not need to check the address. The HC11 then makes a frame with a D of 4 (busy token received, proceed) and B1's containing source and destination address. This frame is then sent onto the network. If the transmitting station receives the 1 token back (signifying an error and that the intended receiver never picked the frame up), it retransmits the same frame and continue sending again.

Table 5.1: Tokens and Their Functions in the Inner HC11's	
Token (binary)	**Functions Performed Because of Tokens**
0000	Test if something to send. No: pass on the free token. Yes: pass on busy token and address information.
0001	Receiver readies the station (including queues) for input. Send 0100 token. Transmitter re-sends busy token and begins transmitting again.
0010	Pass it on. Originating station puts its address to byte 1. If station gets it back, sends free token.
0011	Present transmitting station clears queues and sends out a busy token.
0100	Inner HC11 never receives it.
0101	Receiver which matches one less than station number on byte 1 sends error message to user then shuts down.
0110	First pass: increment byte 1, then store as address. Second pass: store as number of stations. Pass on to next station.
0111	Receivers: ready for input. Transmitter: begin sending data.
1000	Transmitter tests bytes with what was sent. If they match, does nothing. If not, sends the 1101 token and the number of bytes to move back, sets error variable, then begins retransmitting.
1001	Receiver stores bytes in input queue, then sends 1000 token.
1010	Receiver adds bytes to queue then sends 1111 token and clears error.
1011	Receiver puts first byte in the input queue, prepares to output to screen, and sends 1000 token.
1100	Transmitter moves pointer back to last byte received back from the receiver, then begins retransmitting using the 1010 token for the first byte. Receiver sets error variable.
1101	Receiver moves its queue pointer back the number of bytes specified in the first B1.
1110	Receiver stores both bytes, readies output to screen, and passes on 1000 token.
1111	Similar to 1000, but also clears error variable.

If the D is a 3 (sent too many bytes or took too long, stop sending), the HC11 clears the queue and sends an error message to the screen (see inner HC11 fault tolerance for more). The HC11 then sends a free token onto the network.

If the D is a 4 (warning token), the HC11 creates a frame. The frame will have the 4 as the token and its address as the first byte. If the sending HC11 receives this frame back, it knows then that the system is operating and sends a free token onto the network.

If the D is a 5 (error, being removed from the network), the HC11 sends a message to the user and then shut down.

If the D is a 6 (start-up token), the HC11 tests to see if it is the first time it has received this frame. If it is the first time, increment the number in the first B1 byte, store it as the station's address, then pass it on to the next station. If it is the second time, store the first B1 byte as the number of stations, then pass it along. If the station is station 0, it will take the 6 token off of the network after the second pass and send out a free token.

If the D is a 7 (broadcast mode), the input queues will be prepared and the HC11 is set to receive a message. The source address (on the first byte of B1) is placed into the queue and the same frame will be sent on to the next station. If the station is the station which sent the 7 token, the outer HC11 removes the token from the network. Meanwhile, the inner HC11 is sending the data.

If the D is an 8 (data received, send next two bytes), the receiving station compares the two bytes in the B1 channel to the bytes sent. If they do not match, the error variable is set causing any frames with a token of 8 to be ignored and the number of bytes to move back is sent on the first B1 byte with an error token (12) followed by the retransmission of the bytes. If they do match, the secondary queue pointer is incremented to point to the next two bytes on the output queue. For more information on the secondary queue pointer, see the fault tolerance section in this chapter.

If the D is a 9 (data transmit two bytes), the data is added to the input queue, then a frame consisting of the two B1 bytes and token 8 is made and sent onto the network.

If the D is a 10 (retransmit upon request of the receiver), the data are added to the input queue and a frame with token 15 and the two bytes is made and sent onto the network. The error byte (described later) is cleared to indicate to the receiver that it can begin listening to the network again.

If the D is an 11 (last frame, first byte only), the receiver queues the first byte of B1, goes to Output to Screen mode, and sends the 8 token frame back to the transmitter.

If the D is a 12 (error at receiver, retransmit), the transmitting station moves back the queue pointer to the secondary pointer and begins retransmitting. The first frame has a token 10 the rest will go back to a token of 9. This token is sent if the B1 and B2 bytes of the frame do not match. See Fault Tolerance section for more details. An error variable is set to indicate that the receiver should ignore any data flowing in until the retransmitted token (10) arrives.

If the D is a 13 (error at transmitter, retransmit), the receiving station moves back the number of bytes in its input queue specified by the first B1 byte. The retransmitted data writes over the old. If a transmitter receives this token, it calculates the number of frames to move back, places that in byte one of B1, stores its secondary queue pointer into its primary queue pointer, then sends out the frame. The error variable is set to indicate to the transmitter that it should no longer check any returning data since the queue pointer is no longer up to the right point.

The 14 token (last frame, both bytes) is similar to the 12 token, except that the receiver takes both of the B1's instead of only the first.

The last token, 15 (retransmitted data received) is sent to the transmitter to indicate that the first frame of retransmitted data was received. This is used by the transmitter to clear out the error variables, which in turn signals the transmitter to start paying attention to the data that are passing through the network again.

5.1.6 Fault Tolerance Software in the Inner HC11

There are several fault tolerance schemes designed into the program of this system. One of these is the ability of the transmitter to check the data as they return. To do this, the sending queue has two tail pointers: One which points to the next character to be sent (primary queue pointer), and one to point at the last character received back from the network. This allows the transmitter to make sure that what was sent is what was received. See Figure 5.2 for a diagram of the queue pointers (bad data received by the sender will be discussed with the outer HC11's). Several of the tokens are designed to allow for retransmission of bad data, thus making sure that the data transmitted matches the data received.

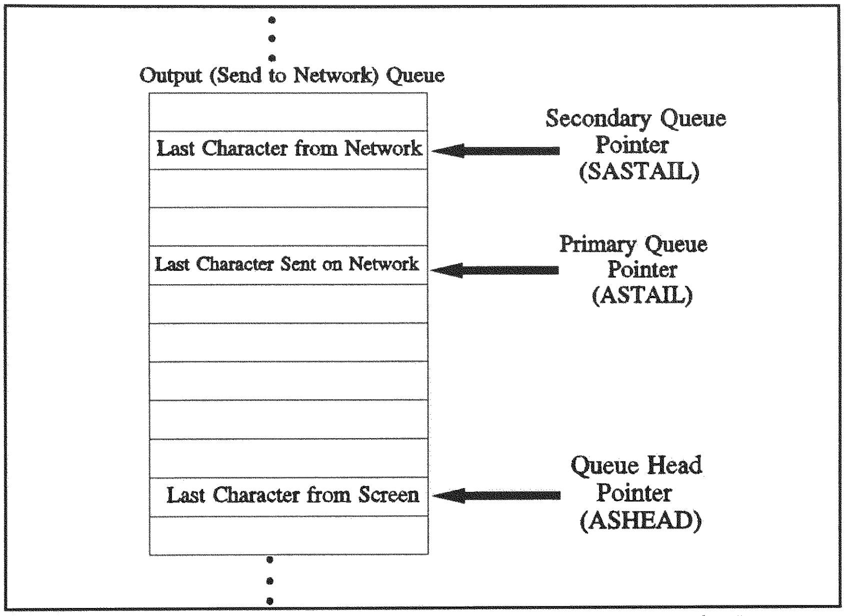

Figure 5.1 Queue Pointers for the Output to Network Queue

The system has three methods of keeping a station from sending a packet which is too large. The first is a count in the transmitter's program. If the sender tries to type in too many characters (more than 256), the HC11 sends a warning message and refuses to take any more characters until it receives a back space, control z, or control x. If this fails, the receiver also counts the number of frames (of good data, retransmitted data not counted) sent. If the transmitter sends more than 128 frames (not counting the busy token frame), the receiver sends the 3 token to force it to stop sending. If both of these fail, a timer interrupt is set at both the receiver and transmitter, which is discussed in section 3.2.2.

Communication stopping completely on the network is handled by a timer on the outer HC11 (described later). The inner HC11 uses the error token to warn the user that he is being taken off of the system and shut down.

Using multiple queues for the input functions (from port C and from the ACIA) can also be considered fault tolerance devices since they allow communication to continue and keep the data from being written over.

5.2 Token Ring Outer HC11 Software

5.2.0 Introduction

The main purposes of the inner HC11s are to communicate with the network user and to output to the network. The functions of the outer HC11 are to act as an intelligent multiplexer, to accept input from the network, as sensors, and as watchdogs to make sure that the network keeps busy. In this section, how these functions are performed on a software level is discussed.

5.2.1 Outer HC11 Input from the Network

The outer HC11's are the direct link of the stations to the network. All input from the network must pass through these stations first. This sets them up as natural editors for the inner HC11's. This saves time and programming space for the inner HC11 by dividing the function tasks, passing some of the most basic tasks onto a piece of hardware which is necessary for multiplexing purposes anyway.

The outer HC11 keeps track of the present function of the station using a function variable. This variable lets the HC11 know if the station is sending, receiving, or doing neither. This is important to know because the function of various tokens depends on this.

The outer HC11 takes the input from one of its two RS-232C ports (which will be discussed later in this chapter) in ISDN frame format. If the HC11 is in Skip mode, it moves to the second byte of the frame to test the first bit of the D nibble (see previous chapter for frame set up). All tokens which only deal with sending and receiving begin with a binary 1, so if the bit is a one, it can be sent straight to the inner HC11 in ISDN format. Since the frame does not concern the station, there is no need to break it down to the D and B channels. If the bit is a 0, the frame contains data which are important to all stations. In this case, and in the case of the station sending and receiving, the outer HC11 gets the D nibble and decides what to do depending on the value.

5.2.2 Output to the Inner HC11

Once the Outer HC11 receives the data from the network, it must pass all necessary information on the inner HC11. As described in the previous section, the HC11's communicate using the C port for data and the A port for handshaking. The outer HC11 uses the token on the D channel to decide what it must pass on to the inner HC11.

The table on the next page describes the function that the HC11 performs for each token. Since most of the more complicated functions are shared by more than one of the tokens, the more detailed descriptions of the special routines focus on the function rather than the token.

For the receiving functions (D = 9, 10, 11, and 14), the B1's are taken from the ISDN frame format and compared with the B2 bytes, which should be a negated echo of the B1's. If they do not match, the retransmit token (12) is sent to the station to be passed on to the transmitter. Transmitting function (D = 8) is also tested. If there is no match, the transmitting error token (13) is passed to the station to be dealt with and sent on to the receiving station. The busy token (D = 1) is also tested and passed on if there is no match.

If the token does not apply to the station (Skip mode and nibble token beginning with a logic 1) the ISDN frame is not broken down to its B1, B2, and D components. In some cases (D = 0, 3, 15 and 12) only the D is needed, so the HC11 will not find the B channels. In most of the other cases, all B's and D's are found with an ISDN break down subroutine.

Table 5.2: Tokens and Their Functions in the Outer HC11's	
Token (binary)	**Functions Performed Because of Tokens**
0000	Pass D to the inner HC11.
0001	Pass D and the first byte B1 (source address) to the inner HC11 and change function to receive if second byte contains station address. If source address is station address, pass D to station. Otherwise, pass on as ISDN frame and change function to skip.
0010	Pass D and first byte B1 to inner HC11. See fault tolerance for more description.
0011	Pass D to inner HC11 and change function to skip if transmitting. Otherwise, pass it on.
0100	If first byte B1 (source address) is station address, set function to sending. If not, pass on as ISDN.
0101	All stations except station which is one less than the address on the receiving station pass it on as ISDN. See fault tolerance section for outer HC11 for description.
0110	First pass: increment byte 1, then store as address. Second pass: store as number of stations. If number of stations = address, then station is 0. Pass D and first byte B1 to station.
0111	Compare address in first byte B1 to station address. Not the same: change function to receive. Same: change function to sending. Pass D and first byte to station.
1000	If sending, then pass D and both bytes B1 to station.
1001	If receiving, pass D and both bytes B1 to station.
1010	If receiving, pass D and both bytes B1 to station.
1011	If receiving, pass D and first byte B1 to station. Change function to skip.
1100	If transmitting, pass D to station.
1101	If receiving, pass D and first byte B1 to station.
1110	If receiving, pass D and both B2 bytes to station. Change function to skip.
1111	If Transmitting, send D and both bytes B1 to station.

Data are transmitted to the station through the C port as described in the section 5.1.5. For a better description of the warning token (D = 2) and the error token (D = 5), see the next section (5.2.2).

5.2.3 Fault Tolerance in the Outer HC11

The most critical part of this experiment is the network's ability to continue functioning even after a station has failed. This, as mentioned before, is accomplished by having each station connected to not only the previous station, but also the station before the previous station. This allows the network to skip over a problem station. This will be accomplished using two timer interrupts as described in the following paragraphs.

The first timer interrupt is based on the amount of time a device has been transmitting. This timer is set to a value based on the number of stations, the baud rate, the maximum number of bits that can be sent (which includes an allowance for up to 20 retransmitted frames) and a small amount of time to allow for processing of the data at each station. The timer is set whenever a station receives a 0, 1, 2, 4, 5, 6 or 7 token (the tokens which indicate that a new message is starting or someone has recognized an error in the system). This means that the station immediately following the station which is sending too much (even if it is not the station presently officially transmitting) is the first to have its timer go off. If the new message token does not arrive by the time the timer goes off, the station then sends a warning token with its address as byte 1 of B1 and listens to both its secondary and primary input. If the warning returns on the primary input before the next timer interrupt, the station sends out a free token. If it hears the warning on the secondary channel, it sends out an error token with its address and starts listening on the secondary input. If it gets nothing but garbage, it assumes that it has an error and shuts itself down by sending out an error token with the next station's address.

Meanwhile, the station immediately following the one sending out the warning listens to its secondary input. If it gets the warning token back from the secondary input and receives the error token from the previous station on its primary input anyway, it remains on the secondary input and increases the address on the warning frame to its address. If it gets a free token on its primary, it goes back to listening to the primary input.

The second timer interrupt will be based on the time between receiving <u>any</u> pair of data bytes . The amount of time between these interrupts is calculated similar to the factor for the timer listed above, but now it is based on only one byte of data. If a byte does not arrive within the specified time, the timer interrupts go off, beginning with the station immediately following the station which received the last byte of data (which is the station which is no longer sending). The steps that are taken are the same as those outlined in the previous paragraphs.

These interrupts are carefully planned so that the station which has the hardware problem is the one removed from the network and to make sure that it is tracked down as soon as possible. The next station is checking its secondary input so that it can determine if the previous station is receiving garbage or is not receiving or if the station before the previous has actually stopped sending or is sending garbage, which is a very important difference.

These two interrupts allow the software to enact the multiplexing fault tolerance which is the central concept behind the network designed in this experiment. However, the outer HC11 also has other fault tolerance capabilities. Foremost of these is the ability to check if the B2 bytes are the negated echo of the B1 bytes. This design is similar to the concept of parity, but it allows each bit to have its own check. These cause a lot of overhead and reduces throughput, but it greatly reduces the chances of a bad piece of data passing undetected. The negation of the data (as opposed to a simple echo) is performed to detect stuck at zero or one faults which would run through all the bytes of the data without otherwise being detected.

5.3 Terminal to Microcontroller Communications

Terminal to microcontroller (68HC11) communications is accomplished using the Terminal program in Windows™. The programs are written on the PC, assembled, and the Terminal software is used to download the code to the memory of the microcontroller. Commands written directly into the software specify where to put the programs in the microcontroller memory.

The Terminal software takes care of all other functions. When the software is running, all data input to the terminal by the keyboard are automatically moved on to the microcontroller. The speed of transfer is determined at connection time by choosing one of the setups available in the Terminal program. Parity, word size, etc. between the terminal and the HC11 can also be chosen at this time.

Chapter 6

Applications of a Fault Tolerant Token Ring Network

6.0 Introduction

This chapter will be a study of systems which could be possible applications of a network similar to the one developed in this research. Included in the study will be a description of what the system is, the benefits of using this network (including the use of the ISDN format), and some general outlines of how the systems will work. The three systems that will be discussed are: A medical network, teacher to students multimedia network, and a general office network.

6.1 Medical Network

6.1.0 Introduction:

Lack of communications is probably one of the leading factors in the problems with the United States' present medical system. Patients, especially the elderly and the employed, have trouble getting to their doctors. Pharmacies have trouble determining if the voice on the other side of the line is really a Doctor or a person trying to get his/her hands on drugs. Ambulance drivers and paramedics are hampered by the time it takes to determine any possible allergies or other illnesses in a patient, and may still end up having to guess in the end and hope that they are not wrong. Communication with insurance companies is slow and involves too much paper work. People on vacation have trouble getting to prescriptions or getting in touch with their Doctors.

The list goes on, but the result is the same: The lack of a strong network between Doctors, hospitals, pharmacies, ambulances, insurance companies, and patients handicaps the American medical system.

6.1.1 The Benefits of the Medical Network

A single network which connects all of the major participants in the medical system (pharmacies, Doctors, patients, etc.) would greatly improve the system.

A patient to Doctor network would allow the patient to get follow ups to tests without having to go back to the Doctor. The ISDN format would allow for several channels, each one carrying a different type of information. For example, with the basic ISDN BRI format, 3 channels are available. One channel could be used for the voice. The second channel could be used to send graphics (X-rays, CAT scans, etc.) to a patients with ISDN format and a computer. The third channel (the D channel) could be used for communications overhead and other data. This third channel can also be used for security. Some type of medical card can be designed which will have specific information about the patient so that the Doctor can be sure that the patient is who s/he claims to be.

Figure 6.1 on the following page illustrates this connection. The patient's computer, card reader/writer, and ISDN phone can be put into ISDN frame format by the network connector. The network connector will also send the message on to the token ring. Since there will be a large number of people on the network and token ring networks get slower with the number of stations, several token ring networks can be tied together. One of the stations on the ring can act as a gateway to another token ring network. It will have the same opportunity to receive and send as any of the other stations on the network.

On the receiving end, the Doctor will have his/her phone and computer. The Doctor's network connector will take the ISDN formatted data and separate it out into phone and computer data, sending it to the appropriate place.

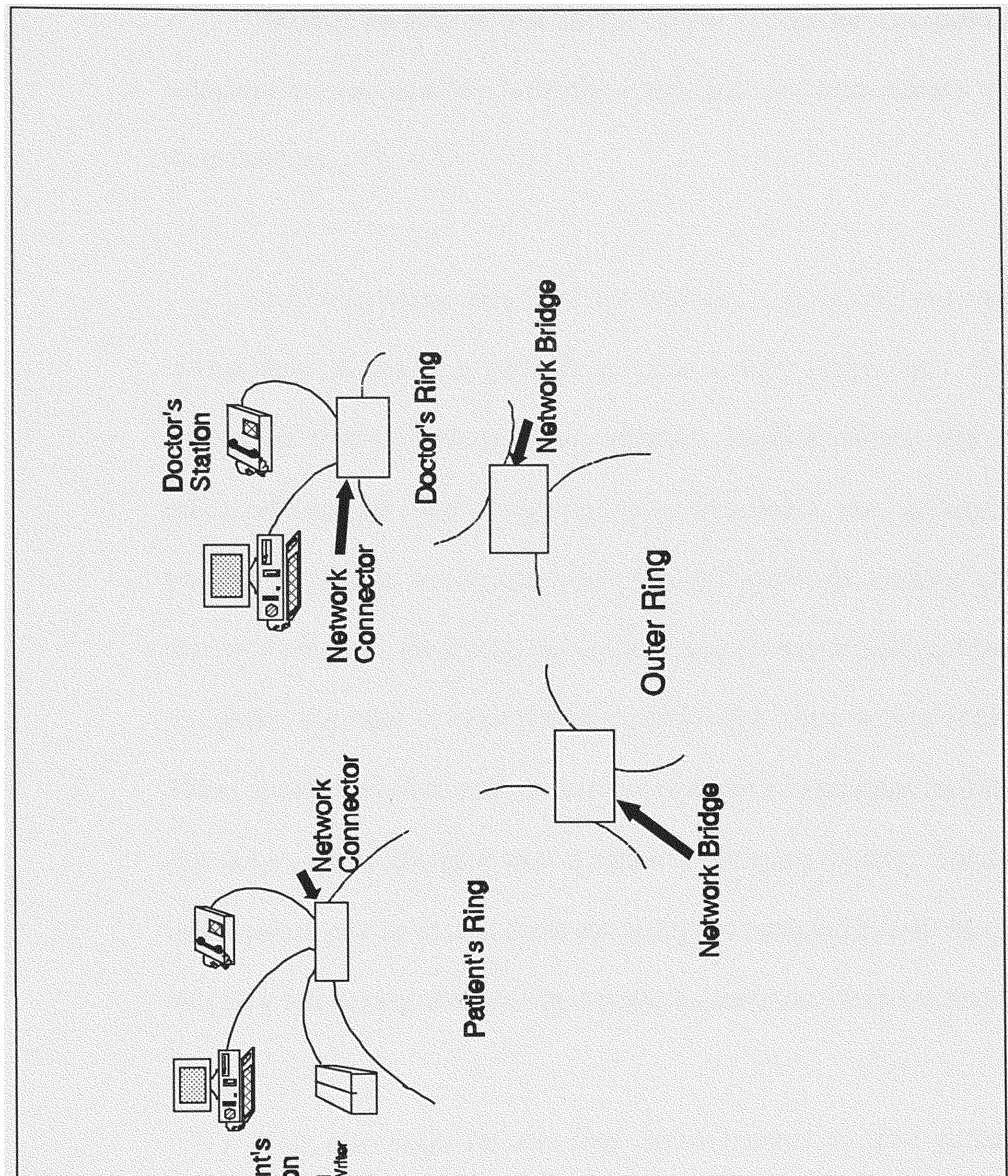

Figure 6.1 Patient to Doctor Network Connection

Another benefit of this ISDN system is that the computer to computer communications will allow the deaf to communicate with their Doctors.

In emergencies, time is of the essence. Ambulance personnel should have access to a patient's records at their finger tips. They could then almost instantly know allergies, medical history, and any other information that is necessary to determine the appropriate treatment. Again, a medical card may be useful in looking up a patient's records. Presently, the medical alert bracelets and necklaces are really the only contact that the paramedics have with a patient. These alerts, however, can not hold very much information, and they can not be easily changed with a patient's changing history. Data on this network can be changed automatically, so paramedics will be sure that they are getting the most up to date information about the patient.

This also helps Doctors keep better records on their patients. Every visit to a hospital, a different doctor, a pharmacy, or, for Doctors with multiple offices, a visit to a different office can be placed directly into the patient's records.

The network can add security to the prescription process. Presently, Doctors call the pharmacies and use a code number read to the pharmacist to leave a prescription. There are three problems with this: Anyone can call a pharmacy, once someone finds out a Doctor's code there is no way to find out that the code has been compromised or track the culprit down, and many pharmacists/clerks are too busy to check the code.

The first benefit of the network designed in this research is that token ring networks are notoriously difficult to break into. Also, the D channel allows Doctors to send specific security codes automatically, adding more security to the system and speeding up the system as a whole. Pharmacists no longer need to take the time to answer the phone because the data can be sent straight to a computer screen. Each time a prescription is refilled, the pharmacist can update the patients records, letting the Doctor know the length of time between refills. For drugs which run out of refills, the pharmacist can send the information and requests for more refills directly to the Doctor without having to make the phone call.

Insurance companies can also be added to the network. This will allow claims to be instantly filed, speeding up the process and decreasing the wait before the Doctor, hospital, or pharmacy is compensated or the patient is reimbursed. Since a large percentage of health care cost comes as a result of administrative costs, making the system more efficient would make health care more affordable.

Figure 6.2 shows the total network connections, including hospitals, Doctors, pharmacies, patients, insurance companies, and ambulances. These will all work similar to the patient to Doctor network described earlier. Again, the large number of stations would probably result in the need for several separate ring networks. Each station is given its own network in the drawing, but this does not necessarily have to be true. For example, a patient may be on the same network as his/her Doctor depending on the location. Hospitals may want their own network so that communications between its labs, Doctors, pharmacy, nurses, etc. may be faster. The ambulance can be connected to the network through a wireless communications scheme, giving them access to the patients records.

To increase security, each station can be given different security clearance. For example, Patient's may not be able to contact a pharmacy by computer at all. Meanwhile, the pharmacy may be able to access only information pertaining to medication, and then only to update prescription dates, not change the medication itself. Security methods can be achieved on the D channel, saving the two B channels for sending data (voice and computer data).

At the present time, most people do not have access to this type of equipment. However, the price of computers and ISDN equipment should eventually drop to a level which will make them affordable to most people.

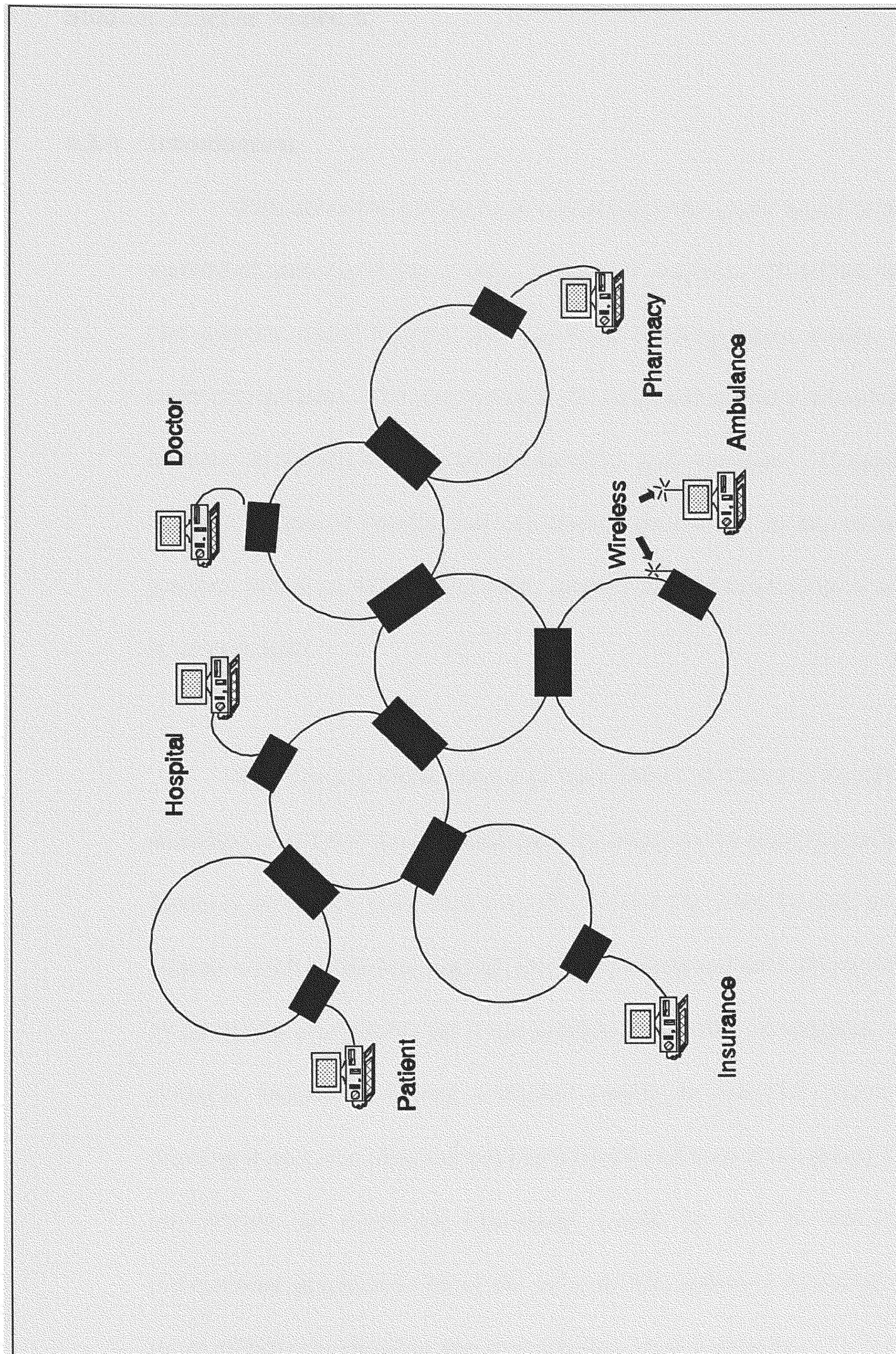

Figure 6.2 Conceptual Integrated Medical Network

6.2 Student-Teacher Network

6.2.0 Introduction

With all of the advances in technology, the chalk board lecture method of teaching is out of date. Computer graphics, CD ROM, laser disc players, sound blasters, etc. can make learning faster, easier, and more entertaining. The combination of visual and auditory stimulus in learning increases student comprehension and retention. Computer learning programs greatly increase student interaction in the learning process, which in turn sparks more interest and concentration in what it being taught.

New classes which consist of the mixture of visual and auditory stimulus, called electronic classrooms, are being developed (Figure 6.3). Students will work from their own PC. These PC's will be able to run lessons which combine computer graphics, sound, and live action video. Multi-media also allows for more graphic illustration of subjects. For example, instead of having a teacher attempt to describe a cube by drawing it on a two dimensional black board and then making odd hand movements, an electronic classroom would be able to use three dimensional graphics to bring the cube to life, spinning it around and more effectively showing the students just what a cube is.

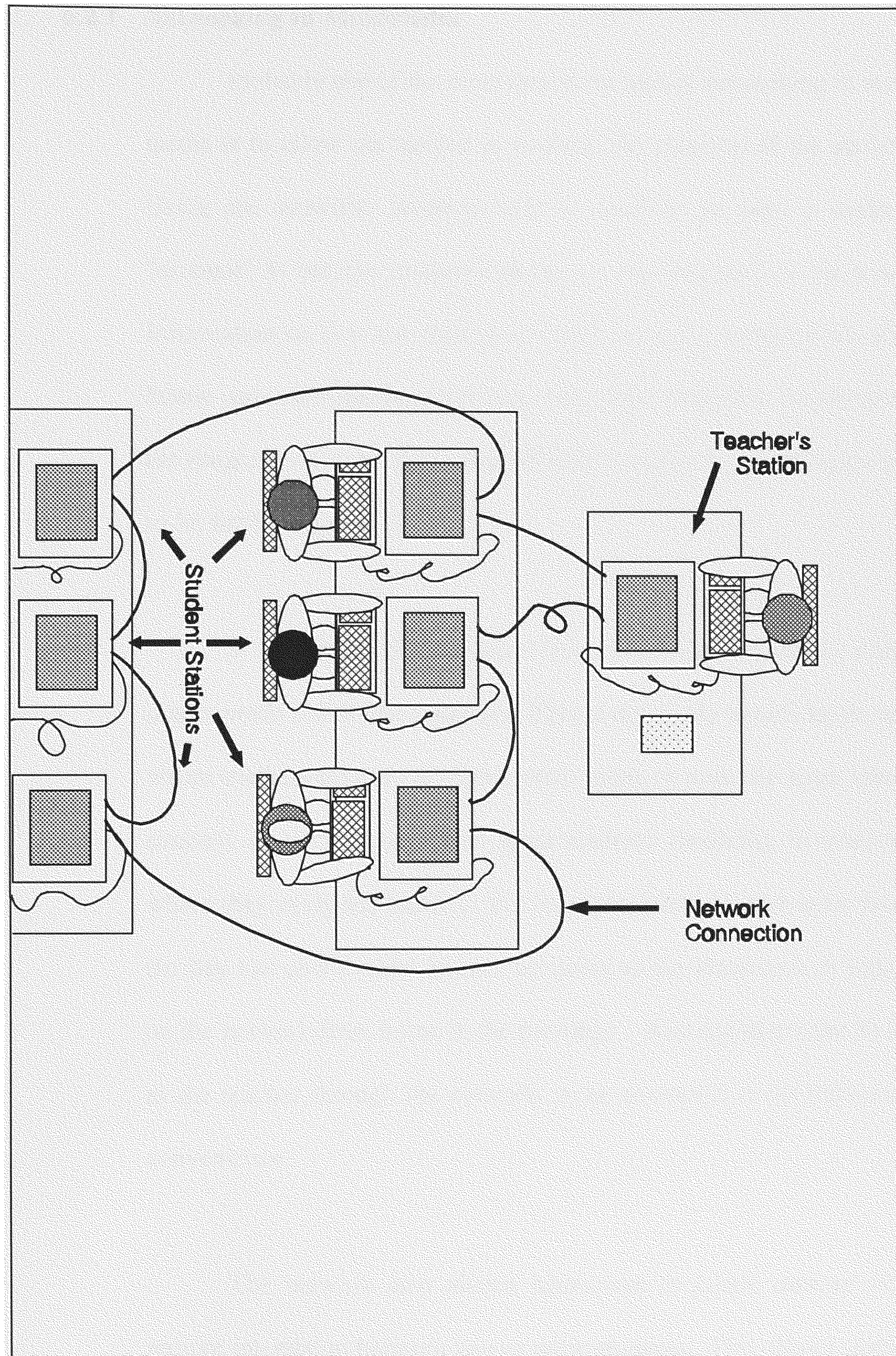

Figure 6.3 Interactive and Intelligent Electronic Classroom

6.2.1 Networking in Multimedia

Probably one of the most important uses of networking in multimedia is to allow the teacher to evaluate the progress of the students. Using the network, teachers will be able to go into a student's "account" to see the student's scores on the exercises in the lesson. Information on how the students are performing in certain areas of the lesson can assist teachers in determining where there are weaknesses in the program, or at the very least indicate to a teacher which topics need to be further explored.

A student could also send messages and questions to the teacher if s/he needs individual attention. This is especially beneficial in cases where teachers may not be in the same room, or even the same state or country, as the student. This gives students flexibility in when and where they study the lessons. For example, students who work during the day can perform the lessons by going to the classroom or logging on the network from home in the evenings . Any questions can be sent to the teacher through the network, to be answered at the instructor's convenience.

The network also allows instructors to create lessons which require interaction between two or more students. This allows students at different locations to interact in solving difficult problems and/or doing problems which focus on increasing their teamwork skills.

Figure 6.4 illustrates the token ring network as a tool in an electronic classroom. Notice that any station can also be a remote station set up on some type of modem. Since most multi-media applications take up so much memory space, a single mainframe with a large amount of memory on the network can reduce the amount of memory required at the individual stations.

Security is important when it comes to students' grades and scores. Again, token ring networks are by nature relatively secure. Along with this, the network can allow students to have individual accounts, protected by passwords, which will store these scores and the present location of the student in the lessons. Teachers can be given special codes which will allow them to gain access to the students' records to evaluate scores on the various exercises.

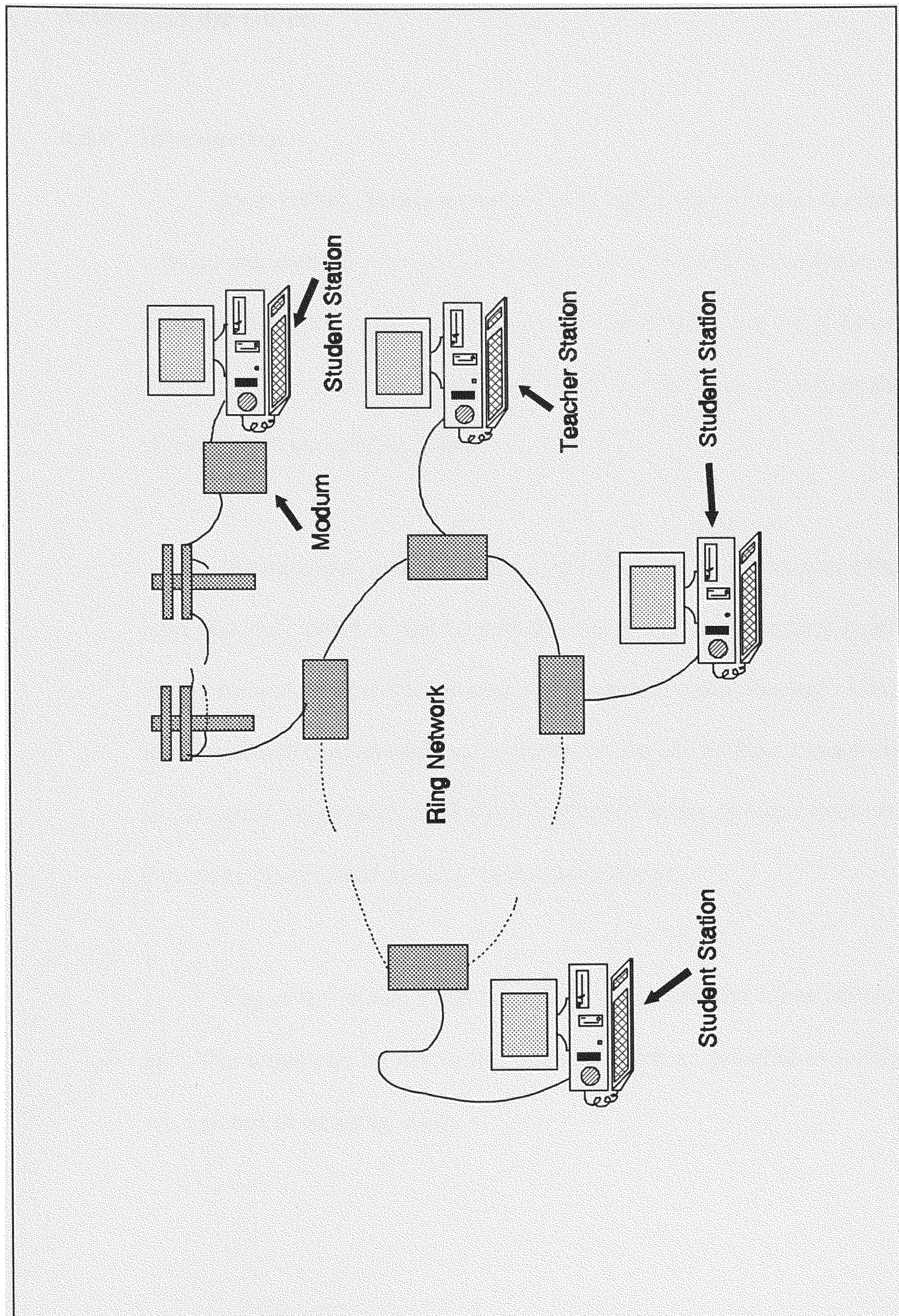

Figure 6.4 Network Diagram for an Electronic Classroom

6.3 Network in the Office

6.3.0 Introduction

Inter-activity between workers in an office environment is often vital to the smooth operation of the office. Traditional methods of sending memos create piles of paper which are wasted and can get lost. The need for quick access to data makes some type of electronic interaction even more desirable.

For example, imagine if one secretary goes on vacation and the boss suddenly needs a paper/letter/etc. that the absent secretary typed on the computer. Searching through the secretary's desk hoping to find the disk which contains the necessary document can be time consuming and fruitless. A network (and a well organized filling system of course) can make finding this missing paper much easier.

Figure 6.5 shows a typical office setting with the token ring set up. The wires can be run in the walls as much as possible to reduce the amount of hanging wires.

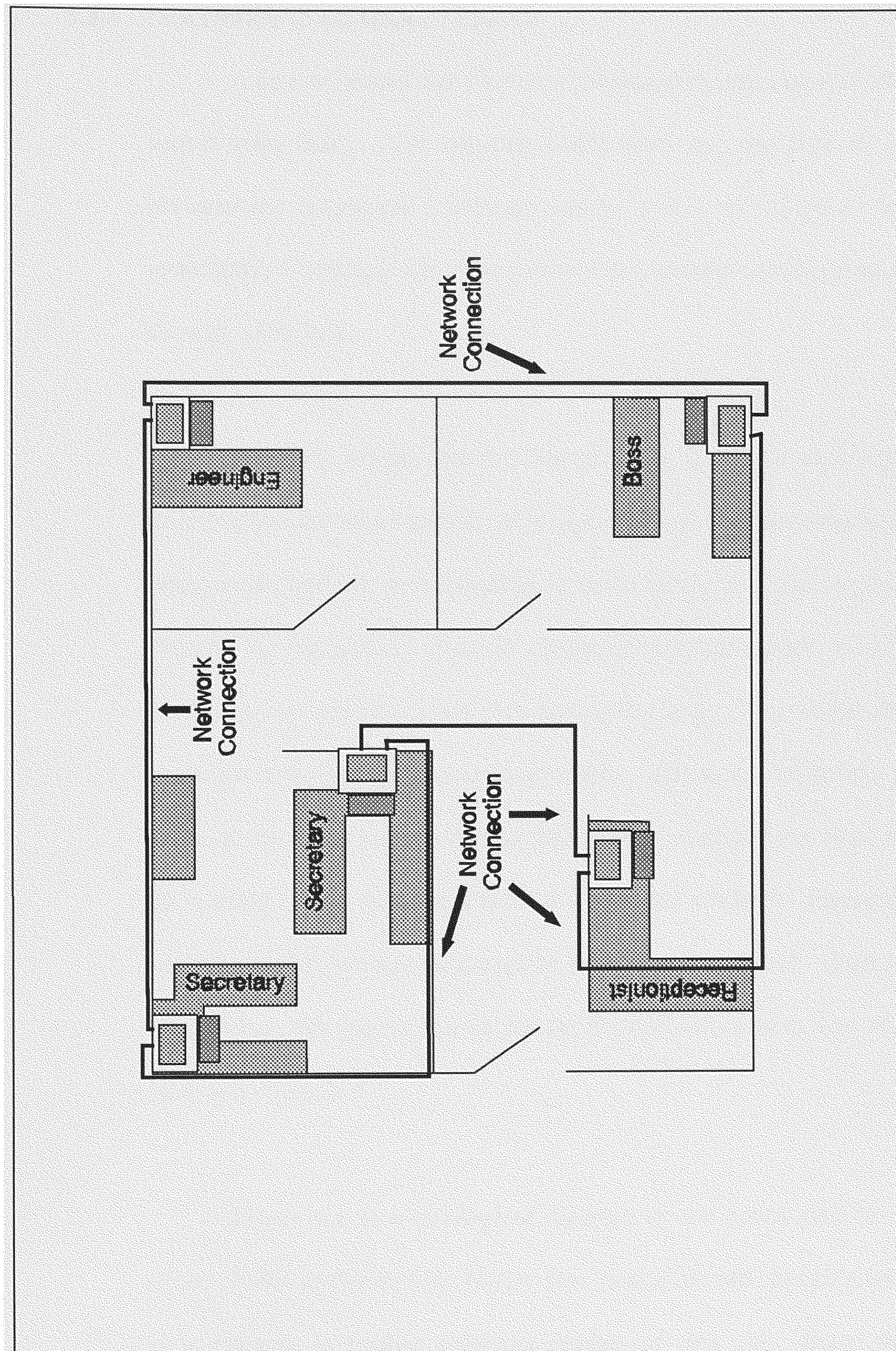

Figure 6.5 Five Station Office Token Ring Network

6.3.1 Description of an Office Network

A fault tolerant token ring network would fit into a typical office format quite easily. The network would allow the free flow of data between the employees. Security can be added as needed to keep unauthorized employees from sensitive documents by creating levels of security clearance.

There are several uses for this network. To begin with, papers or letters written by a secretary or receptionist can be sent to one of the other employees for proof reading without having to waste paper by printing the document. Next, an electronic mailing system would be easy to set up after the network is already in place. This would allow messages to be passed between employees. Third, several people can work on the same paper simultaneously. For example, the Boss may be working on the economic side of a proposal while the Engineer is putting together the technical aspects of the same proposal. Once they are finished, the papers can be put on the network, sent to one of the two, and combined.

Figure 6.6 is a conceptual diagram of the token ring for the office shown in Figure 6.5. Notice that there is an added station which is a gateway to further networks outside of this one. Since these employees probably work together more often than with other employees outside of this immediate office or someone outside of the company all together, they can be given their own sub-network with which to work. This will speed up their communications to each other and isolate them from other sub-networks in case some part of the system elsewhere in the company goes down.

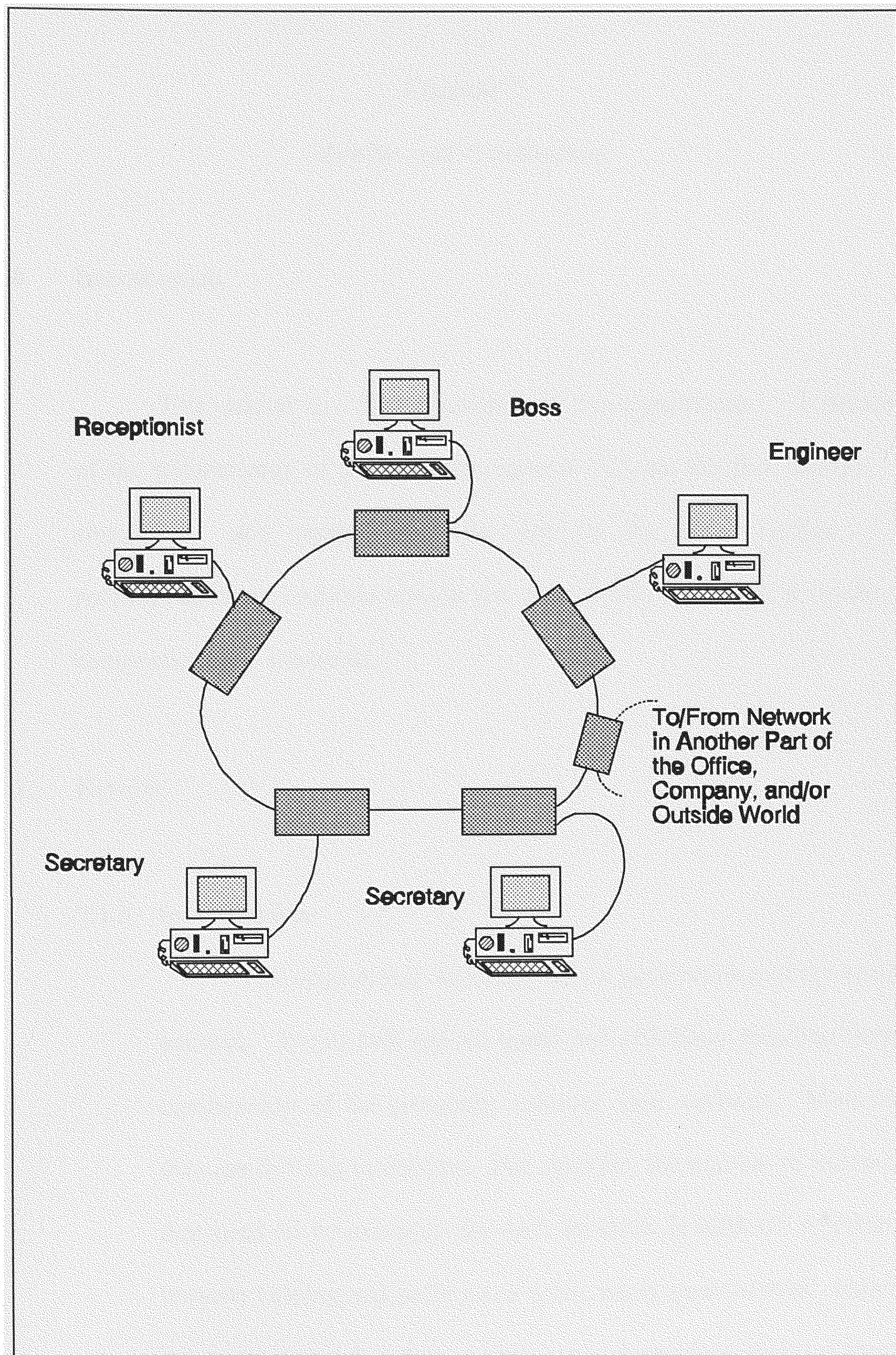

Figure 6.6 Diagram of an Office Token Ring Network

Chapter 7

Results and Conclusions

7.0 Introduction

This chapter is a final discussion of this experiment. A discussion of the performance of the resulting network follows the first section. The conclusions and recommendations are in the final section. The recommendations and conclusions are divided into hardware, software, and communications protocols.

7.1 Results

7.1.0 Introduction

This section is a discussion of the performance of the designed network. Topics will include speed and efficiency resulting from the combination of the protocols, software, and hardware. Most of the data are difficult to confirm. For example, the number of instructions that need to be executed for each function is difficult to determine because looping and polling can create many uncertainties. However, the numbers used in this section came from close study of the final program.

7.1.1 Speed of the System

The HC11 ports were set to 9600 baud. This is the maximum baud for reliable communications between the PC and the ACIA. Greater speeds begin to increase the corruption of data sent. To remain consistent, the entire system, including the host ports, was set to 9600 baud. Greater baud rates can be achieved through changes in registers and hardware, but this baud rate served its purpose.

The RS-232 ports send one byte of data at a time. Along with this data, the ports send a start bit and a stop bit. This increases each transmission to ten bits, and each frame to sixty bits. Therefore, one frame takes approximately 6.25 ms (60bits/9600 baud) to move from one HC11 to another on the network. Therefore, a frame would have n*6.25 ms, where n is the number of stations, of travel time to work its way through the network. For example, the network built has three stations. This means that a frame would take 18.75 ms of travel time to move through the network.

This, however, does not include execution time at each station. Each type of token can mean different things to on outer HC11. In fact, each frame of data can mean different things to different stations on the network. For example, if station 1 is sending to station 3, station 2 should be in skip mode. Station 1 must pull the data from the queue, put it into ISDN frame format, and send it. Meanwhile, it must also listen to the network waiting for the acknowledgement, test the returned data for errors, then either retransmit or move on to the next test. Station 3 must decode the D channel, decode the B channels, test for errors, queue the data (including incrementing counters), insert the appropriate acknowledgement token, put the frame into ISDN frame format, and send the frame. Station 2, however, will pass the data straight through its system as an ISDN frame format. This means that station 2 has a shorter execution time than the other two stations.

Table 7.1 shows some approximate execution times. The number of instructions executed is an approximation based on careful study of the program. An average of 4 cycles per command was chosen because much of the program consists of loads and stores of immediate numbers (2 cycles each), load and store effective address (4 cycles each) decrements and increments (mostly 2 but sometimes 3 cycles, depending on the register), branches (3 cycles), jumps and returns (6 cycles and 5 cycles), and load index registers (5 cycles). The HC11 has a clock of about 2 MHz.

Table 7.1: Execution Time for Various Modes		
Mode of Operation	**Number of Instructions to Execute**	**Execution Time (msec)**
Skip Station	200	0.40
Transmit	450	0.90
Receive Last Bytes	600	1.20

Notice that all of these times are quite a bit smaller than the 6.25 ms transmission time. This shows that the stations are done processing and sending out the frame before the next frame arrives at the station. This results in overlapping of execution time and transmitting time, which reduces the amount of time that the system sees for sending a packet of information. This means the frames themselves take the same amount of time, but with the overlap the packet as a whole moves faster.

Table 7.2 shows the approximate transmitting time, from the transmitting station back to the transmitting station, of frames/packets through the system. This is based on packet size, three stations, and the assumption that only the first and the last execution times are seen by the network due to the function overlap. Free tokens are assumed to take about 200 instructions to execute.

Table 7.2: Time to Traverse the Network	
Circumstance	**Time (msec)**
Free Token	19.95
1 Frame in a packet Broadcast	22.05
1 Frame in a Packet Non-broadcast	21.25
129 (maximum) Frame Packet	934.85

The broadcast mode takes more time because all stations are receiving. Non-broadcast has at least one station which is skipped. Notice that the transfer time of a whole packet is much less than the sum of its parts. The table below illustrates the overlap and how it cuts down the transmission time.

Table 7.3: Illustration of Communications Overlap on a 9 Frame Packet			
Time	**Frame at Station 1**	**Frame at Station 2**	**Frame at Station 3**
1	F1	--	--
2	F2	F1	
3	F3	F2	F1
4	F4	F3	F2
5	F5	F4	F3
6	F6	F5	F4
7	F7	F6	F5
8	F8	F7	F6
9	F9	F8	F7
10		F9	F8
11			F9

Notice that although there are 9 frames sent to 3 stations, only 11 transmission times are seen rather than 27. This is a great saving in time.

7.1.2 Efficiency of the Network

For every 6 bytes (frame) sent, only 2 of those bytes are actual data. This means that only 33% (16/48) of the information sent is data. When the start and stop bits of the RS-232 ports are added, this percentage drops even further to about 27% (16/60). This gives an actual data rate of only 2592 bps. For free token frames, the percentage drops even more since the only necessary data is on the D channel (4/60 = 7%).

However, these numbers are a little misleading. For each frame, 20 bits can be considered used for fault tolerance (20/60 = 33%). Since one of the focuses of this research is on fault tolerance, this loss is worthwhile. The only actual, unused bits are the twelve overhead bits of the ISDN frame and the stop and start bits of the RS-232, which totals twenty four of the sixty bits, or about 40%. Unfortuneatly, to keep with ISDN frame format and RS-232, this large percentage must be endured.

For any network, efficient use of the line is desirable. The least efficient time for the ring network is when the token is traveling through the network. In this case, the line is used for 18.75 msec in the 19.95 msec it takes the token to travel around the network. This leaves the line idle for 1.2 msec, or about 6% of the time. When the percentage of actual data rate is figured in, this drops the network down to carry worthwhile data about 6.2% of the time.

The network becomes more efficient when a packet is being sent. As seen earlier, the overlap makes the network move smoothly so that there is almost always something on the line. The larger the packet, the more the line idle percentage drops toward 0.0%. This means that the RS-232 line is busy nearly all of the time. When the percentage of useful data is factored in, this gives a total efficiency of about 60% if fault tolerant bits are factored in, or about 20% if they are not.

7.2 Conclusions and Recommendations for Future Study

7.2.1 Hardware Recommendations and Conclusions

In the early stages of this research, the choice of microcontroller to use to implement the network design was a major decision. In the end, materials available became the strongest argument for the 68HC11EVB board. As the work progressed and more was learned about the board, the pros and cons of the 68HC11EVB became more apparent.

The 68HC11EVB microcontroller is very flexible and performed well in this experiment. Debugging programs can be difficult, especially for the outer 68HC11EVB. With no direct contact to the outer, the limited debugging abilities of the board became very apparent. However, researchers would probably have problems debugging an outer board regardless of the microcontroller.

One feature on a microcontroller that a future researcher may want to look for is more RS-232 ports on a single board. A microcontroller with four RS-232 ports would be ideal. These ports would supply the two inputs from the network, one output to the network, and one port to communicate with the computer. This would eliminate the need for an outer microcontroller and save the programmer quite a bit of work.

More advanced microcontrollers, such as the MVME133 VME board (68020 microprocessor based mircocomputer), would add a few benefits to the system. The 68020VME is a 32-bit microcontroller with a coprocessor. With its eight 32-bit data registers and 7 address registers, the 68020 processor is more flexible than the 68HC11 with its two 8-bit accumulators and two sixteen bit index registers. These added registers would make the programming much easier and faster. By giving some of the variables which are most often used their own accumulators, the expanded register set would reduce the need for loads and stores.

The benefits of the register set of the 68020 is outweighed by its limited number of ports. Also, the HC11 instruction set is good enough to perform all of the functions for this experiment. It is doubtful that an expanded instruction set would simplify the program.

Due to limits in communication rates with PC's, a 9600 baud rate was chosen for all communications. With the limited number of stations and small packets allowed, this baud rate was enough. However, a network designed with a higher baud rate would be desirable for larger networks. Transmission time through the RS-232 ports takes most of the packet communication time. Compared to the transmission time, the instruction execution was very small. This means that the frames could be sent continuously and processing could occur between frames. However, at higher baud rates this may not be true. Eventually, the HC11 would not be able to keep up with the transmissions and delays would have to be added. Therefore, for higher baud rates a faster microprocessor is desirable.

However, for this experimental network, the 68HC11EVB turned out to be a good choice. Increased knowledge of microcontrollers in general and the 68HC11EVB in particular resulted from implementing the network. Efficient use of ports, instruction code, and registers were three of the benefits. Also, the importance in timing for networks became apparent. Theoretical research often downplays the importance of timing. Building the network showed just how important timing actually is.

The hardware used to skip a station proved adequate to the situation. The reverser and the multi-port devices performed even better than expected. The outer HC11 worked well as a software driven multiplexer, again justifying its use in this project.

7.2.2 Protocol Conclusions and Recommendations

With an experimental network such as this, a mixture of many protocols can be achieved. In this case, the ISDN frame format was combined with the token ring protocol and a very modified version of the FDDI timing protocol. This turned out to be a good mixture. It supplied all the necessary components to result in a successful network.

Since one of the purposes of this research was to learn more about the protocols used, these recommendations concentrate more on applying the protocols chosen in different ways than in applying other protocols.

The ISDN frame format fulfilled its purpose well. The three channels naturally supplied capability for a large variety of tokens (D channel) and fault tolerance in the form of echoing the data (the B2 channel). The overhead made it a little inefficient, but the benefits far outweighed this.

One of the benefits of ISDN is the ability to send to two separate stations simultaneously. This ability could also be applied here. The frame could be broken up into its two channels: B1 going to one station, B2 to another. For example, byte one of B1 in an ISDN frame would carry data from station 1 to station 3. Byte two of B1 could be used for the negated echo. If less fault tolerance is acceptable, it could be used for a second byte of data. Meanwhile, B2 could perform the same function for a communication being sent from station 2 to station 1. The software for this would be much more complicated. The D channel would have to be split if it were to still hold token information. Two bits would be used for one channel, two for the other. This would limit the variety of tokens allowed and further decrease fault tolerance. However, the ability of ISDN to send several communications simultaneously would be exploited.

In designing and building the token ring, its strengths and weaknesses had to be completely explored. Designing the software for the network illustrated that the token ring is much more complicated than it sounds. Keeping track of the token and keeping the ring running is difficult. However, once it is running, it is very naturally fault tolerant. The concept of the token proved to be an effective method for keeping the network from dissolving into a free-for-all. The circular nature of the ring creates a serious risk of a byte or frame of data endlessly looping through the network. This makes programming more difficult, but the single direction of the token ring compensated for this. The final organized network made the effort worthwhile.

The timing methods which are very loosely based on the FDDI protocol should prove very effective in isolating and overcoming faults. These methods keep the time that a network is down due to a station fault to a minimum. Also, they are effective in keeping any one station from tying up the line. Any station which does not abide by the rules can be removed from the network.

These three protocols turned out to be complementary. By taking what was necessary, an effective protocol was developed and implemented. Without a strong protocol, the entire network would have fallen apart.

7.2.3 Software Conclusions and Recommendations

This project supplied an excellent opportunity to learn how to write code which must be a strong combination of speed and efficient use of memory.

The limited RAM space of the HC11 and the amount of room that was necessary for the queues made limiting the program a major objective. Loops and sub routines were used wherever possible to decrease the amount of repeated code. The programs turned out to be well within the RAM limits of the HC11. The assembly coding kept the code efficient. However, for future study, programming in a higher level language such as C would be less time consuming.

As with any other network, execution speed of the code is important. Most of the code was limited to commands which took very few cycles. Loading and storing accumulators are probably the two most common functions, taking up between three and five cycles depending on addressing used.

Interrupts were kept to a minimum to make the program easier to follow. However, for faster networks interrupt schemes for the ACIA and the host ports of both the inner and outer HC11's should be implemented. This would increase the speed of execution. The timing interrupt would be an excellent method for minimizing the threat of failure. Future research should study the possibility of implementing this timing method. With the 68HC11, the host and ACIA ports have different handshaking requirements; therefore, implementing the timing proved too difficult. Other microcontrollers may not have this problem.

Other than this, the code worked well. The programming was a good learning experience. For example, converting to and from ISDN was a wonderful exercise for using the logical shift commands. It was also a good exercise in alternating between the A and B accumulators and the D accumulator.

7.2.4 Final Remarks

This project turned out to be a good exercise in programming, applying the HC11 to a situation, designing and building a network, and designing fault tolerant schemes.

The greatest benefits came from studying a network as close as was required. This exercise increased awareness of how complicated and fascinating networking methods are.

Also, the great amount of time spent considering fault tolerant options made the project worthwhile. The experiment led to greater understanding of both the traditional methods of fault tolerance and applying software and hardware (such as the multi-port and the timer interrupts) to come up with new methods.

Bibliography:

1. Ali, M. Ifran, "Frame Relay in Public Networks," IEEE Communications Magazine, January, 1991.

2. Byrne, William R., Et Al, "Evolution of Metropolitan Area Networks to Broadband ISDN," IEEE Communications Magazine, January, 1991.

3. Bux, Werner, "Token-Ring Local Area Networks and Their Performance," Proceedings of the IEEE, Vol. 77, No. 2, February 1989.

4. Child, Jeffrey, "RISC Chips Continue Conquest of Embedded Realm," Computer Design, Vol. 32, May, 1992.

5. Corbin, M.J., and Jones, J.G. "Method for Monitoring Faults in a Control Computer Using Band-Limiting Filters," IEE Proceedings, Vol. 137, Pt. D, No. 2, March, 1990.

6. Fowler, David, "Perpetual Networks," Byte, August 1991.

7. Gaitonde, S. S., Jacobson, D. W., and Pohm, A. V., "Bounding Delay on a Multifarious Token Ring Network," Communications of the ACM, Vol 33, No. 1, January, 1990.

8. Gaonkar, Ramesh, Microprocessor Architecture, Programming, and Applications, Second Edition, Macmillan Publishing Company, New York, 1989.

9. Huwicz, Michael, "FDDI: Not Fastest But Still Fit," Datamation, Vol. 39, Apr. 1, 1993.

10. Johnson, Barry, Design and Analysis of Fault Tolerant Digital Systems, Addison Wesley Publishing Company, New York, 1989.

11. Jung, W. Y., "Analysis of Throughput and Delay of a High-Speed Slotted Ring Based on Lumped Modeling," IEEE Transactions on Communications, Vol. 40, May 1992.

12. Kubat, Peter, "A Digital Circuit Performance Analysis for tandem Burst-Error Links in an ISDN Environment," IEEE Transactions on Communications, Vol. 37, Oct. 1989.

13. Motorola, HC11, M68HC11 Reference Manual, Prentice Hall, Englewood Cliffs, New Jersey, 1989.

14. Motorola, Microprocessor, Microcontroller, and Peripheral Data, Vol. 1, Motorola Inc., USA, 1988.

15. Motorola, M68HC11EVB Evaluation Board User's Manual, First Edition, Motorola, Inc., USA, 1986.

16. Qu, Yaoshuang and Landweber, Lawrence, "Parallelring: A Token Ring LAN with Concurrent Multiple Transmissions and Message Destination", IEEE Transactions on Communications, Vol. 40, No. 4, April 1992.

17. Silios, Charles B., Jr., "An Approximate Method for the Performance of Playthrough Rings," IEEE Transactions on Computers, Vol. 41, Sept 1992.

18. Stallings, William, Data and Computer Communications, Second Edition, Macmillan Publishing Company, New York, 1988.

19. Stallings, William, ISDN and Broadband ISDN, Second Edition, Macmillan Publishing Company, New York, 1992.

20. Subbarao, Wunnava, 16/32-Bit Microprocessors 68000/68010/68020 Software, Hardware, and Design Applications, Macmillan Publishing Company, New York, 1991.

21. Weiss, Ray, "68HC11 Adapts to 3.3V Designs," EDN, Vol. 37, May 7, 1992.

22. Wilson, Dave, "Is RISC or DSP Best for Your Application?", Computer Design, Vol. 32, April, 1992.

Appendix:

Program Listings

```
*  This is the program for the inner HC11

*  Setting constants for various port addresses
PORTA            EQU   $1000
PORTCL           EQU   $1005
ACIA             EQU   $9800
PIOC             EQU   $1002
DDRC             EQU   $1007
DDRD             EQU   $1009
BAUD             EQU   $102B
SCCR1            EQU   $102C
SCCR2            EQU   $102D
SCSR             EQU   $102E
SCDR             EQU   $102F

*  Setting constants for various ASCI character codes

LF               EQU   $0A
CR               EQU   $0D
QT               EQU   $18
ET               EQU   $1A
BS               EQU   $08
CL               EQU   $19

*  Setting starting address for the program

                 ORG   $C000

*  Initializing stack pointers

                 LDS   #STACK
                 LDX   #QUEUE1
                 STX   AITAIL
                 STX   AIHEAD
                 LDX   #QUEUE2
                 STX   ASTAIL
                 STX   ASHEAD
                 LDX   #QUEUE3
                 STX   INCHEAD
                 STX   INCTAIL
                 LDX   #QUEUE4
                 STX   TOSCREENT
                 STX   TOSCREENH
                 LDX   #ISDN
                 STX   ISDNHEAD
```

```
        STX   ISDNTAIL

* Iniatializing ACIA port for poling

        LDAA  #$16
        STAA  ACIA

* Clearing counters and function indicators

        LDAA  #$00
        STAA  BROADCAST
        STAA  SENDING
        STAA  ERROR
        STAA  ERRMESS
        STAA  OUT2SC
        STAA  SCOUNT
        STAA  SENDSING
        STAA  SCCR1
        STAA  DESTINATION
        STAA  NOSTATS
        STAA  INFRAME
        STAA  COUNT

* Setting other ports

        STAA  DDRC
        STAA  PIOC
        LDAA  #$40
        STAA  PORTA
        LDAA  #$0A
        STAA  SCCR2
        LDAA  #$30
        STAA  BAUD

* Setting basic address to FF to start

        LDAA  #$FF
        STAA  MYADDRESS

* Setting the stations to receiving to start

        LDAA  #$01
        STAA  RECEIVING
```

```
*  Startup routine ... Asks user if they are station 0

STARTUP         EQU   *
                JSR   CLEARSC
                LDY   #MSG1
                LDAA  #$01
                STAA  ERRMESS
                JSR   WAITUP
                BRA   ANSWER


WAITUP          EQU   *
                JSR   PRINTSCREEN
                LDAA  ERRMESS
                BNE   WAITUP
                RTS

*  Clears the screen

CLEARSC         EQU   *
                PSHA
                PSHB
                LDY   #CLEAR
                LDAA  #$01
                STAA  ERRMESS
                JSR   WAITUP
                PULB
                PULA
                RTS

*  Waiting for an answer ... Yes (Y) station 0, or No (N) not station 0.
*       Otherwise, wait for a valid answer

ANSWER          EQU   *

                LDAA  ACIA
                BITA  #$01
                BEQ   ANSWER
                LDAA  ACIA+1
                JSR   CLEARSC
                CMPB  #'Y'
                BNE   ISNO
                CLR   MYADDRESS
                LDAA  #$06
                STAA  DC
```

```
                CLR   BYTE1
                LDAA  #$FF
                STAA  BYTE2
                JSR   MAKEFRAME
                LDAA  #$01
                STAA  SENDSING
                BRA   MAIN

ISNO            EQU   *
                CMPB  #'N'
                BEQ   MAIN
                BRA   STARTUP


* Main loop

MAIN            EQU   *

* Test for input from the ACIA (from user)

                LDAA  ACIA
                BITA  #$01
                BNE   INACIA1

* Test for input from port C (outer HC11)

                LDAA  PIOC
                BITA  #$80
                BNE   INC1

* Test to see if sending a single byte of data

                LDAA  SENDSING
                BNE   SENDONE

* Test to see if sending a packet

                LDAA  SENDING
                BITA  #$01
                BNE   SEND1

* Test to see if in the process of sending an error message

                LDAA  ERRMESS
                BNE   SENDERROR
```

```
*  Test to see if presently sending to the screen from the network. If not,
*       all tests done so return  to beginning of loop.  Otherwise, output
*       next character to screen if screen buffer not full.

                LDAA  OUT2SC
                BEQ   MAIN
                LDX   TOSCREENT
                LDAB  0,X
                LDAA  ACIA
                BITA  #$02
                BEQ   MAIN
                STAB  ACIA+1
                CMPB  #CR

*  If output to screen is carriage return, add a line feed

                BEQ   ADDLINE
                INX

*  Move output to screen queue, if it is empty, signal no longer outputing,
*       otherwise, continue loop

                STX   TOSCREENT
                CPX   TOSCREENH
                BNE   MAIN
                CLR   OUT2SC
                BRA   MAIN

* Subroutine to output a line feed

ADDLINE         EQU   *
                LDAB  #LF
                STAB  0,X
                BRA   MAIN

*  Sends program to sub routine which deals with an input from the ACIA

INACIA1         EQU   *
                JSR   INACIA
                BRA   MAIN

*  Sends program to sub routine which deals with an input from C, then
*       sends message to outer HC11 that data has been received
```

```
INC1            EQU   *
                JSR   INC
                LDAA  #$40
                STAA  PORTA
                BRA   MAIN
```

* Sends program to sub routine which sends a single byte on to network

```
SENDONE         EQU   *
                JSR   SCHAR
                BRA   MAIN
```

* Continues sending an error message

```
SENDERROR       EQU   *
                JSR   PRINTSCREEN
                BRA   MAIN
```

* Sends program to sub routine which deals with continuing output of a packet

```
SEND1           EQU   *
                JSR   SEND
                BRA   MAIN
```

* Sub routine to clear the screen

```
CLSC            EQU   *
                LDAA  #$01
                STAA  ERRMESS
                LDY   #CLEAR
                RTS
```

* Sub routine which gets an input from the ACIA (User)

```
INACIA          EQU   *
                LDAB  ACIA+1
```

* Test input

```
CHECK           EQU   *
                CMPB  #QT
                BEQ   DUMP
                CMPB  #ET
                BEQ   BEGINSEND
                CMPB  #BS
```

```
                BEQ    REMOVE
                CMPB   #CL
                BEQ    CLSC
                LDAA   SENDING
                BEQ    DESADD

*  If not a special character, test counter.  Too many characters, send error
*       message.  Otherwise,  add to queue, increment appropriate counters,
*       echo character to screen, and return to main program.

ADDTOQ          EQU    *
                LDAA   SCOUNT
                CMPA   #$FF
                BEQ    ERRORMESS
                LDX    AIHEAD
                STAB   0,X
                INX
                STX    AIHEAD
                INC    SCOUNT

PSC             EQU    *
                JSR    ECHO
                CMPB   #CR
                BEQ    LINEF
                RTS

*  Add a line feed it it is a character return being echoed

LINEF           EQU    *
                LDAB   #LF
                BRA    PSC

*  If it is a back space, clear previous character, echo to screen decrement
*       counters, and return to main program

REMOVE          EQU    *
                LDX    AIHEAD
                DEX
                STX    AIHEAD
                DEC    SCOUNT
                JSR    ECHO
                RTS
```

```
ECHO            EQU  *
                LDAA    ACIA
                BITA  #$02
                BEQ   ECHO
                STAB  ACIA+1
                RTS

* If too many characters (>256) input, ready approp. error message

ERRORMESS       EQU  *
                LDY   #MSG2
                LDAA    #$01
                STAA    ERRMESS
                JSR     PRINTSCREEN
                RTS

* If it is the first character in a message, it is the destination address.

DESADD          EQU  *
                SUBB    #$30
                STAB  DESTINATION
                LDAA    #$01
                STAA    ERRMESS
                LDY   #CLEARIN
                LDAA    #$08
                STAA    SENDING
                RTS

* If it is the quit character, clear input queue, counters, and function
*       indicator, then return to main program

DUMP            EQU  *
                LDX   AITAIL
                STX   AIHEAD
                CLR   SCOUNT
                CLR   SENDING
                CLR   DESTINATION
                RTS

* If it is exit character, set the system up to send the message

BEGINSEND       EQU  *
```

```
*  If nothing to send, ignore the exit command

                    LDX   AIHEAD
                    CPX   AITAIL
                    BEQ   NOTDONE

*  If the previous message is still being sent, ignore the exit command

                    LDX   ASHEAD
                    CPX   ASTAIL
                    BNE   CANT

*  If everything is okay, clear counter and move ACIA input queue to
*       network output queue and vice versa

                    CLR   SCOUNT
                    LDX   AIHEAD
                    STX   TEMPA1
                    LDX   AITAIL
                    STX   TEMPA2
                    CPX   #QUEUE1
                    BNE   SWAP
                    LDX   #QUEUE2

*  Swaps queue pointers

SWAP                EQU   *
                    STX   AIHEAD
                    STX   AITAIL
                    LDX   TEMPA1
                    STX   ASHEAD
                    LDX   TEMPA2
                    STX   ASTAIL
                    STX   SASTAIL

*  Indicating ready to send in function indicator

                    LDAA  #$04
                    STAA  SENDING
                    RTS

*  Send error message is can't send yet

CANT                EQU   *
                    LDY   #MSG3
```

```
                LDAA    #$01
                STAA    ERRMESS
                JSR     PRINTSCREEN

NOTDONE         EQU     *
                RTS


*  Make first frame with approp. D, source address in B1, and destination
*       address in B2.  If destination address greater than the number of
*       addresses, send as broadcast.

MAKEFIRST       EQU     *
                CLR     RECEIVING
                LDAA    #$01
                STAA    DC
                LDAA    MYADDRESS
                STAA    BYTE1
                LDAA    DESTINATION
                CMPA    NOSTATS
                BLT     MAKEBUSY
                LDAB    #$07
                STAB    DC
                LDAA    #$02
                STAA    BROADCAST
                LDAA    #$FF

*  Make frame, then return to main program

MAKEBUSY        EQU     *
                STAA    BYTE2
                JSR     MAKEFRAME
                RTS

*  If function indicator indicates that in the process of sending, send character

SEND            EQU     *
                LDAB    SENDING
                BITB    #$F6
                BEQ     SCHAR

*  Testing where station is in sending

DEALTOKEN       EQU     *
```

```
* If there is still more of an ISDN frame to send, send next byte

                LDX   ISDNTAIL
                CPX   ISDNHEAD
                BNE   SENDIT

* If last byte of free token just sent, clear sending function

                CMPB  #$03
                BEQ   THATSALL

* Otherwise, make free token and prepare to send it

                CLR   DC
                LDAA  #$FF
                STAA  BYTE1
                STAA  BYTE2
                JSR   MAKEFRAME
                LDAA  #$06
                STAA  SENDING
                RTS

* Send next byte of ISDN frame

SENDIT          EQU   *
                LDAB  0,X
                INX
                STX   ISDNTAIL
                JSR   HOST
                RTS

* When host RS-232 ready, send a byte

HOST            EQU   *
                JSR   DELAY
                LDAA  SCSR
                BITA  #$80
                BEQ   HOST
                STAB  SCDR
                RTS

* After last byte of free token sent, clear out everything and return to main

THATSALL        EQU   *
                CLR   SENDING
```

```
                LDX   #ISDN
                STX   ISDNTAIL
                STX   ISDNHEAD
                RTS

* This sub routine sends ISDN frames from the Host RS-232

SCHAR           EQU   *

* Load and send next byte in ISDN queue

                LDX   ISDNTAIL
                LDAB  0,X
                JSR   HOST
                INX
                STX   ISDNTAIL
                CPX   ISDNHEAD
                BNE   OUTTAHERE

* If there is no more bytes in the ISDN frame, test if sending single frame

                LDAA  SENDSING
                BNE   ALLDONE

* If not sending single frame, test if resending

                LDAA  SENDING
                CMPA  #$09
                BEQ   MOVINGBACK

* If not resending, get next character and put into B1

                LDX   ASTAIL
                LDAB  0,X
                STAB  BYTE1
                INX
                CPX   ASHEAD
                BEQ   LASTONE

* If not last character in queue, get next character in queue and put in B2

                LDAB  0,X
                STAB  BYTE2
                INX
                CPX   ASHEAD
```

```
                BEQ   LASTTWO

* If that is not the last character, set D to 9

                LDAB    #$09
                STAB DC
                BRA   ALLSET

* If moving back, set continue sending

MOVINGBACK      EQU   *
                LDAA    #$01
                STAA    SENDING
                JSR   MAKEFRAME
                RTS

* If last character, set B2 to FF and D to 11 (indicating last character being sent)

LASTONE         EQU   *
                LDAB    #$FF
                STAB BYTE2
                LDAB    #$0B
                STAB DC
                LDAB    #$05
                STAB SENDING
                BRA   ALLSET

* If last two characters, set D to 14

LASTTWO         EQU   *
                LDAB    #$0E
                STAB DC
                LDAB    #$05
                STAB SENDING

* Make frame with B1, B2, and D set in above routines, then return to main

ALLSET          EQU   *
                STX   ASTAIL
                JSR   MAKEFRAME
                RTS

* When finished sending a single frame, reset pointers and return to main

ALLDONE         EQU   *
```

```
                LDX   #ISDN
                STX   ISDNTAIL
                STX   ISDNHEAD
                CLR   SENDSING

OUTTAHERE       EQU   *
                RTS

* This routine makes B1, B2 and D into an ISDN frame format

MAKEFRAME       EQU   *

* First byte of ISDN

                LDX   #ISDN
                STX   ISDNTAIL
                STX   ISDNHEAD
                LDAA  BYTE1
                LDAB  #$00
                LSRD
                LSRD
                ORAA  #$40
                JSR   STORE

* Second byte

                TBA
                LDAB  DC
                BITB  #$08
                BEQ   OTHER
                ORAA  #$3A
                BRA   FBYTE2

OTHER           EQU   *
                ORAA  #$2A

FBYTE2          EQU   *

* Getting B1' to put into second byte of B1

                LDAB  BYTE1
                EORB  #$FF
                LSRA
                LSLD
                JSR   STORE
```

```
*  Third byte

                    ORAB   #$01
                    TBA
                    JSR    STORE

*  Fourth byte
                    LDAA   DC
                    LSRA
                    ORAA   #$01
                    LDAB   BYTE2
                    LSRD
                    LSRD
                    TBA
                    JSR    STORE


*  Fifth byte
                    LDAA   BYTE2
                    CLRB
                    LSRD
                    LSRD
                    TBA
                    LDAB   DC
                    ANDB   #$02
                    BNE    OTHER2
                    ORAA   #$28
                    BRA    FBYTE5


OTHER2              EQU    *
                    ORAA     #$38

FBYTE5              EQU    *
                    LDAB     BYTE2
                    EORB     #$FF
                    LSRA
                    LSRA
                    LSRA
                    LSLD
                    LSLD
                    LSLD
                    JSR      STORE

*  Last byte
```

```
        LDAA   DC
        BITA  #$01
        BEQ   OTHER3
        ORAB   #$07
        BRA   FBYTE6

OTHER3  EQU  *
        ORAB   #$05

FBYTE6  EQU  *
        TBA
        JSR   STORE
        RTS

*  Routine to store byte into ISDN frame

STORE   EQU  *
        LDX  ISDNHEAD
        STAA   0,X
        INX
        STX   ISDNHEAD
        RTS

*  This routine takes care of an input from C

INC     EQU  *

*  If A is 0, then input it in B1, B2, and D form

        LDAA   PORTA
        BITA  #$01
        BNE   ISDNFORM

*  Load first byte, store it in D, and signal to outer HC11 that data is read

        LDAA   PORTCL
        CLR   PORTA
        STAA   DC
        BNE   COMP1

*  If D = 0 (free token), test if there is something to send

        LDAA  SENDING
        CMPA  #$04
        BNE   ELSE1
```

```
* If there is, make the first frame and begin sending

                LDAA  #$01
                STAA  SENDING
                JSR   MAKEFIRST
                RTS

* Otherwise, send free token back on to network

ELSE1           EQU   *
                LDAA  #$FF
                STAA  BYTE1
                STAA  BYTE2
                JSR   MAKEFRAME
                LDAA  #$01
                STAA  SENDSING
                RTS

* If A not 0, then the data being sent is in ISDN form.  Send it straight on,
*       not for this station

ISDNFORM        EQU   *
                JSR   ISDNFORM1
                RTS

COMP1           EQU   *

* If MSB of D is a one, go to upper test (8 thru 15)

                BITA  #$08
                BNE   GOCOMP8

                CMPA  #$01
                BNE   COMP3

* If D = 1 (busy token), test if this station is sending.  If it is, an error has occured

                LDAA  SENDING
                CMPA  #$01
                BEQ   NOONEGOT

* If this station is not sending, it must be a message for this station.
*       Load acknowledgement (D = 0), this station's address into B2, and
*       get source address from port C (put into input from network stack).
```

```
                LDAA  #$04
                STAA  DC
                LDAA  MYADDRESS
                STAA  BYTE2
                JSR   GETBYTE
                STAA  BYTE1
                ADDA  #$30
                JSR   ADDINC
                RTS

* If noone got the first busy token, send it again and begin resending
*       everything sent so far.

NOONEGOT        EQU   *
                LDAA  MYADDRESS
                STAA  BYTE1
                LDAA  DESTINATION
                STAA  BYTE2
                LDAA  #$01
                STAA  DC
                LDAA  #$09
                STAA  SENDING
                LDX   SASTAIL
                STX   ASTAIL
                RTS

* Adds source address taken in from C port into appropriate Queue, sets
*       appropriate counters, and sends acknowledgement

ADDINC          EQU   *
                LDX   INCHEAD
                STAA  0,X
                INX
                STX   INCHEAD
                CLR   INFRAME
                JSR   MAKEFRAME
                LDAA  #$01
                STAA  SENDSING
                STAA  RECEIVING
                RTS

* Goes to compare upper D's

GOCOMP8         EQU   *
                JMP   COMP8
```

```
* Takes a byte in from port C

GETBYTE         EQU   *
                LDAA  #$40
                STAA  PORTA

WAIT            EQU   *
                LDAA  PIOC
                BITA  #$80
                BEQ   WAIT
                LDAA  PORTCL
                CLR   PORTA
                RTS

* If D = 3, sent too many bytes.  Clear sending queue, send error message
*       to screen, send free token

COMP3           EQU   *
                CMPA  #$03
                BNE   COMP5
                LDY   #MSG4
                LDAA  #$01
                STAA  ERRMESS
                LDX   ASTAIL
                STX   ASHEAD
                STX   SASTAIL
                CLR   DC
                LDAA  #$01
                STAA  SENDSING
                LDAA  #$FF
                STAA  BYTE1
                STAA  BYTE2
                LDX   ISDNHEAD
                CPX   ISDNTAIL
                BNE   WAITALLDONE
                CLR   SENDING
                JSR   MAKEFRAME
                RTS

* If ISDN frame has not been completely sent, wait until it is finished
*       to stop sending

WAITALLDONE     EQU   *
                LDAA  #$09
                STAA  SENDING
```

```
                RTS

* If D = 5, send error message to screen, then shut down from network

COMP5           EQU  *
                CMPA  #$05
                BNE   COMP6
                LDY   #MSG5

WAITDONE        EQU  *
                JSR   PRINTSCREEN
                LDAA  ERRMESS
                BNE   WAITDONE
                END

* If D = 6, startup token

COMP6           EQU  *
                CMPA  #$06
                BNE   COMP7
                JSR   GETBYTE
                STAA  BYTE1
                LDAA  MYADDRESS
                BEQ   ZERO
                LDAA  #$02

ZERO            EQU  *
                ORAA  COUNT
                BEQ   LTCOMP
                CMPA  #$01
                BNE   NXCOMP

* If this is the second time station 0 has gotten the startup token, send free token

                LDAA  #$00
                STAA  DC
                LDAA  #$FF
                STAA  BYTE1
                STAA  BYTE2
                BRA   ALLDONE2

* If this is the first time a non-0 station has gotten startup, get B1 from
*       port C, increment the value, and put that number into address.

NXCOMP          EQU  *
```

```
                CMPA    #$02
                BNE   LTCOMP
                LDAA    BYTE1
                INCA
                STAA    BYTE1
                STAA    MYADDRESS
                BRA   ALLDONE2

*  If second time non-0 or first time for station 0, get B1 and store it as
*       the number of stations

LTCOMP          EQU  *
                LDAA    BYTE1
                INCA
                STAA    NOSTATS

*  Make a frame and send it out

ALLDONE2        EQU  *
                JSR    MAKEFRAME
                LDAA    #$01
                STAA    COUNT
                STAA    SENDSING
                RTS


*  If D = 7, broadcast mode.  Ready for input

COMP7           EQU  *
                LDAA    #$FF
                STAA    BYTE2
                JSR    GETBYTE
                STAA    BYTE1
                ADDA    #$30
                JSR    ADDINC
                LDAA    #$01
                STAA    BROADCAST
                RTS

*  If D = 8, then get B1 and B2 that were received and compare to what was sent

COMP8           EQU  *
                BITA  #$04
                BNE   GOCOMP12
                CMPA    #$8
```

```
                BNE   COMP9
                LDAA   ERROR
                BNE   RET2
                JSR   GETBYTE
                STAA   B1
                JSR   GETBYTE
                STAA   B2

COMPARETOSEND      EQU   *
                LDX   SASTAIL
                LDAA  0,X
                CMPA  B1
                BNE   NOTSAME
                INX
                STX   SASTAIL
                CPX   ASHEAD
                BEQ   SENDFREE
                LDAA  0,X
                CMPA  B2
                BNE   NOTSAME
                INX
                STX   SASTAIL
                CPX   ASHEAD
                BNE   RET2

*  Once the last byte or two bytes come back, send free token

SENDFREE        EQU   *
                LDAA   #$03
                STAA   SENDING
                CLR   BROADCAST

RET2            EQU   *
                RTS

GOCOMP12        EQU   *
                JMP   COMP12

*  If sent does not equal received, send the error token and begin resending
*       from last byte received back

NOTSAME         EQU   *
                LDAA   #$0D
                STAA   DC
                JSR   MOVEBACK
```

```
                RTS

MOVEBACK        EQU   *
                LDX   SASTAIL
                LDAA    0,X
                STAA    BYTE1
                INX
                LDAA    0,X
                STAA    BYTE2
                INX
                STX   ASTAIL
                INC   ERROR
                LDX   ISDNHEAD
                CPX   ISDNTAIL
                BNE   NOTYET
                JSR   MAKEFRAME
                RTS

* If in the middle of sending a frame, wait before resending

NOTYET          EQU   *
                LDAA    #$09
                STAA    SENDING
                RTS

* If this station is sending, broadcast, and a 9 is received, treat it as an 8

IMSEND          EQU   *
                LDAA    #$08
                BRA   COMP8

* If D = 9, data being sent.  Get B1 and B2 from port C.  If not in error mode,
*       add to Queue.  If broadcast, send data out with D = 9, if not, send
*       data out with D = 8.

COMP9           EQU   *
                CMPA    #$09
                BNE   COMP10
                LDAA    BROADCAST
                CMPA    #$02
                BEQ   IMSEND
                JSR   GETBYTE
                STAA    BYTE1
                JSR   GETBYTE
                STAA    BYTE2
```

```
                LDAA    #$08
                ADDA    BROADCAST
                STAA    DC
                LDAA    ERROR
                BNE     IGNORE
                JSR     ADD2Q
                JSR     MAKEFRAME
                LDAA    #$01
                STAA    SENDSING
                RTS

IGNORE          EQU     *
                RTS

*  Adds data from port C to input Queue.  If too many characters sent,
*       send error token

ADD2Q           EQU     *
                LDX     INCHEAD
                LDAA    BYTE1
                STAA    0,X
                INX
                LDAA    BYTE2
                STAA    0,X
                INX
                STX     INCHEAD
                LDAA    INFRAME
                INCA
                STAA    INFRAME
                CMPA    #$80
                BNE     IGNORE
                LDAA    #$03
                STAA    DC
                RTS

*  If D = 10, data resent.  Begin getting data again.  Send okay token.

COMP10          EQU     *
                CMPA    #$0A
                BNE     COMP11
                JSR     GETBYTE
                STAA    BYTE1
                JSR     GETBYTE
                STAA    BYTE2
                CLR     ERROR
```

```
          LDAA   #$0F
          STAA   DC
          JSR    ADD2Q
          JSR    MAKEFRAME
          RTS

* If D = 11, sent last byte

COMP11    EQU    *
          LDAA   BROADCAST
          CMPA   #$02
          BEQ    IMSEND2
          JSR    GETBYTE
          STAA   BYTE1
          CLR    BYTE2
          JSR    READYOUT
          RTS

* If I'm sending broadcast, test what was sent to what was received.

IMSEND2   EQU    *
          CLR    BROADCAST
          JSR    GETBYTE
          STAA   B1
          JMP    COMPARETOSEND

* If this station is receiving, begin outputing to screen

READYOUT  EQU    *
          CLR    RECEIVING
          JSR    ADD2Q
          LDAA   #$01
          STAA   OUT2SC
          STAA   ERRMESS
          LDY    #CLEARREC
          LDX    INCHEAD
          STX    TOSCREENH
          LDX    INCTAIL
          STX    TOSCREENT
          CPX    QUEUE3
          BNE    MOVE
          LDX    QUEUE4

* Move around queues to begin inputing again, and send okay token
```

```
MOVE            EQU   *
                STX   INCTAIL
                STX   INCHEAD
                LDAA    BROADCAST
                BNE   FINISH
                LDAA    #$08
                STAA    DC

*  Make appropriate token and send it out.

FINISH          EQU   *
                JSR    MAKEFRAME
                LDAA    #$01
                STAA    SENDSING
                CLR   INFRAME
                CLR   BROADCAST
                RTS

*  D = 12, begin retransmitting

COMP12          EQU   *
                CMPA    #$0C
                BNE   COMP13
                LDAA    RECEIVING
                BNE   SE
                LDAA    #$01
                STAA    ERROR
                STAA    SENDSING
                JSR    MAKEFRAME
                RTS

SE              EQU   *
                LDAA    #$0A
                STAA    DC
                JSR    MOVEBACK
                RTS


*  If D = 13, error.  If receiving, move back and cut out appropriate characters.

COMP13          EQU   *
                CMPA    #$0D
                BNE   COMP14
                LDAA    RECEIVING
                BEQ   TRANS
```

```
                JSR   GETBYTE
                LDAB  INFRAME
                STAA  INFRAME
                LDX   INCTAIL

MB              EQU   *
                DEX
                DECA
                BNE   MB
                STX   INCTAIL
                LSR   INFRAME
                SUBB  INFRAME
                STAB  INFRAME
                RTS


*  If transmitting, get a new frame together with the number of bad frames that
*       have been sent

TRANS           EQU   *
                LDD   ASTAIL
                SUBD  SASTAIL
                STAB  BYTE1
                LDAA  #$FF
                STAA  BYTE2
                LDX   SASTAIL
                STX   ASTAIL
                LDAA  #$09
                STAA  SENDING
                RTS

*  If D = 14, broadcast, and I am sending, stop sending and treat as D = 8

IMSEND3         EQU   *
                LDAA  #$08
                CLR   BROADCAST
                JMP   COMP8

*  If D = 14, get last 2 bytes and finish

COMP14          EQU   *
                CMPA  #$0E
                BNE   COMP15
                LDAA  BROADCAST
                CMPA  #$02
```

```
        BEQ   IMSEND3
        JSR   GETBYTE
        STAA  BYTE1
        JSR   GETBYTE
        STAA  BYTE2
        JSR   READYOUT
        RTS

*  If D = 15, clear error and begin transmitting again

COMP15          EQU   *
                CLR   ERROR
                LDAA  #$08
                JSR   COMP8
                RTS

*  If data coming is in ISDN form, get 6 bytes, put into ISDN queue, and send it on

ISDNFORM1       EQU   *
                LDAA  #$40
                STAA  PORTA
                LDAA  #$06
                STAA  COUNT
                LDX   #ISDN
                STX   ISDNHEAD
                STX   ISDNTAIL

GETIN           EQU   *
                LDAA  PIOC
                BITA  #$80
                BEQ   GETIN
                LDAA  PORTCL
                CLR   PORTA
                JSR   STORE
                LDAA  #$40
                STAA  PORTA
                DEC   COUNT
                BNE   GETIN
                LDAA  #$01
                STAA  SENDSING
                RTS

*  Outputing message to screen

PRINTSCREEN     EQU   *
```

```
                PSHA
                PSHB
                LDAB    0,Y
                CMPB    #'$'
                BNE   CHECK1
                CLR   ERRMESS

GOON            EQU   *
                PULB
                PULA
                RTS

*  Test to see if ACIA ready for next output character

CHECK1          EQU   *
                LDAA    ACIA
                BITA  #$02
                BEQ   GOON
                STAB  ACIA+1
                INY
                PULB
                PULA
                RTS

*  Short delay

DELAY           EQU   *
                PSHA
                LDAA    #$10
DELAY1          EQU   *
                DECA
                BNE   DELAY1
                PULA
                RTS

*  Variables:

MYADDRESS       RMB  1
SENDING         RMB  1
ERRMESS         RMB  1
OUT2SC          RMB  1
SCOUNT          RMB  1
SENDSING        RMB  1
COUNT           RMB  1
RECEIVING       RMB  1
```

```
DC              RMB  1
BYTE1           RMB  1
BYTE2           RMB  1
DESTINATION     RMB  1
NOSTATS         RMB  1
INFRAME         RMB  1
B1              RMB  1
B2              RMB  1
ERROR           RMB  1
BROADCAST       RMB  1

*  Pointers:

AITAIL          RMB  2
AIHEAD          RMB  2
ASTAIL          RMB  2
ASHEAD          RMB  2
INCHEAD         RMB  2
INCTAIL         RMB  2
TOSCREENT       RMB  2
TOSCREENH       RMB  2
ISDNHEAD        RMB  2
ISDNTAIL        RMB  2
TEMPA1          RMB  2
TEMPA2          RMB  2
SASTAIL         RMB  2

*  Setting up bytes necessary for clearing screen:

CLEAR           FCB   $1B
                FCC   '[2J'
                FCB   $1B
                FCC   '[3;0H$'

*  Clear receiving side:

CLEARREC        FCB   $1B
                FCC   '[16;0H'
                FCB   $1B
                FCC   '[K'
                FCB   $1B
                FCC   '[17;0H'
                FCB   $1B
                FCC   '[K'
                FCB   $1B
```

```
                FCC   '[18;0H'
                FCB   $1B
                FCC   '[K'
                FCB   $1B
                FCC   '[19;0H'
                FCB   $1B
                FCC   '[K'
                FCB   $1B
                FCC   '[20;0H'
                FCB   $1B
                FCC   '[K'
                FCB   $1B
                FCC   '[21;0H'
                FCB   $1B
                FCC   '[K'
                FCB   $1B
                FCC   '[22;0H'
                FCB   $1B
                FCC   '[16;0H$'

* Clearing input side:

CLEARIN         FCB   $1B
                FCC   '[3;0H'
                FCB   $1B
                FCC   '[K'
                FCB   $1B
                FCC   '[4;0H'
                FCB   $1B
                FCC   '[K'
                FCB   $1B
                FCC   '[5;0H'
                FCB   $1B
                FCC   '[K'
                FCB   $1B
                FCC   '[6;0H'
                FCB   $1B
                FCC   '[K'
                FCB   $1B
                FCC   '[7;0H'
                FCB   $1B
                FCC   '[K'
                FCB   $1B
                FCC   '[8;0H'
                FCB   $1B
```

```
        FCC  '[K'
        FCB  $1B
        FCC  '[9;0H'
        FCB  $1B
        FCC  '[3;0H$'

*  Setting up bytes for messages:

MSG1            FCB  $1B
        FCC  '[10;20H'
        FCC  'AM I STATION 0 (Y OR N)? $'
MSG2            FCB  $1B
        FCC  '[10;20H'
        FCC  'YOU HAVE REACHED THE END OF THE
BUFFER.$'
MSG3            FCB  $1B
        FCC  '[10;20H'
        FCC  'CAN'T SEND MESSAGE YET.$'
MSG4            FCB  $1B
        FCC  '[10;20H'
        FCC  'SENT TOO MANY BYTES.    ABORTING
TRANSFER.$'
MSG5            FCB  $1B
        FCC  '[10;20H'
        FCC  'SYSTEM ERROR.  BEING REMOVED FROM THE
NETWORK.$'


*  Setting up room for queues:

QUEUE1      RMB  257
QUEUE2      RMB  257
QUEUE3      RMB  257
QUEUE4      RMB  257

* setting up room for ISDN queue:

ISDN        RMB  256

*  Setting up stack for system:

STACK       RMB  1
```

```
*  This is the program for the outer HC11's

*  Inititializing port addresses:

PORTA           EQU   $1000
PORTCL          EQU   $1005
ACIA            EQU   $9800
PIOC            EQU   $1002
DDRC            EQU   $1007
DDRD            EQU   $1009
BAUD            EQU   $102B
SCCR1           EQU   $102C
SCCR2           EQU   $102D
SCSR            EQU   $102E
SCDR            EQU   $102F

*  Initializing constants

LF              EQU   $0A
CR              EQU   $0D

*  Starting program at $C000 in memory

                ORG   $C000

*  Initializing stack pointers

                LDS   #STACK
                LDX   #ISDN
                STX   HEAD
                STX   TAIL

*  Initializing ports:

                LDAA #$16
                STAA ACIA

*  Clearing variables and initializing ports:

                LDAA #$00
                STAA SECOND
                STAA NOSTATS
                STAA SCCR1
                STAA PIOC
                STAA PORTA
```

```
                STAA COUNT
                STAA DC

* Initializing function to skip:

                LDAA #$02
                STAA FUNCTION

* Initializing ports:

                LDAA #$0D
                STAA SCCR2
                LDAA #$30
                STAA BAUD

* Clearing ISDN bytes:

                LDAA #$FF
                STAA DDRC
                STAA BYTE1A
                STAA BYTE1B
                STAA BYTE2A
                STAA BYTE2B


* Main loops

MAIN            EQU  *

* Test and input from network

                LDAA ACIA
                BITA #$01
                BEQ  MAIN
                LDAB ACIA+1
                LDX  HEAD
                STAB 0,X
                INX
                STX  HEAD
                LDAA COUNT
                INCA
                STAA COUNT

* If all 6 bytes of a frame have not been received, loop back to main
```

```
                CMPA #$06
                BNE   MAIN
                LDAA FUNCTION
                CMPA #$2

*  If sending or receiving, send test input

                BNE   TESTD1
                LDX   TAIL
                INX

*  If first bit of D = 1 and skip mode, loop back to main.  Otherwise, test D

                LDAA 0,X
                DEX
                BITA  #$10
                BEQ   TESTD1
                JSR   SENDISDN
                BRA   MAIN

*  Get D, test it, etc.  Return by cleaning out queue and counter and return to main
*        loop

TESTD1          EQU   *
                JSR   TESTD
                LDX   #ISDN
                STX   HEAD
                STX   TAIL
                CLR   COUNT
                BRA   MAIN

*  Pull D out of ISDN frame and test it to do appropriate functions

TESTD           EQU   *
                JSR   GETD
                LDAA DC
                BITA  #$08
                BEQ   ZERO
                JMP   COMP8

ZERO            EQU   *
                BITA  #$04
                BNE   COMP4

*  If D = 0, Send D only to inner HC11 and put self into skip mode
```

```
COMP0           EQU   *
                CMPA #$00
                BNE   COMP1
                JSR   SENDCHAR
                LDAA #$02
                STAA FUNCTION
                RTS

*  If D = 1, get B1 and B2, then test to see if it is to this station

COMP1           EQU   *
                CMPA #01
                BNE   COMP3
                JSR   GETBS
                LDAA BYTE2A
                CMPA MYADDRESS
                BNE   DUMPIT
                JSR   COMPARE
                LDAA DC
                CMPA #$01
                BNE   DUMPIT2

*  If message is to this station, send D and source address to inner HC11

                JSR   SENDCHAR
                LDAA BYTE1A
                JSR   SENDCHAR
                CLR   FUNCTION
                RTS

*  If not to this station, go into skip mode

DUMPIT          EQU   *
                LDAA #$02
                STAA FUNCTION

*  And send to inner HC11 in ISDN frame format

DUMPIT2         EQU   *
                JSR   SENDISDN
                RTS

*  If D = 3, too many characters sent.  If sending, go into skip mode and pass
*       D on to inner HC11.
```

```
COMP3           EQU   *
                LDAA FUNCTION
                CMPA #$01
                BNE  NOTMINE
                LDAA DC
                JSR  SENDCHAR
                LDAA #$02
                STAA FUNCTION
                RTS

* If D = 4, test to see if this station is sending the message. If yes, set to
*      sending mode. If no, pass it on

COMP4           EQU   *
                CMPA #$04
                BNE  COMP5
                JSR  GETBS
                LDAA BYTE1A
                CMPA MYADDRESS
                BNE  NOTMINE
                LDAA #$01
                STAA FUNCTION
                RTS

* If message does not concern this station, pass it on as an ISDN frame

NOTMINE         EQU   *
                JSR  SENDISDN
                LDAA #$02
                STAA FUNCTION
                RTS

* If D = 5, go to receiving mode

COMP5           EQU   *
                CMPA #$05
                BNE  COMP6
                CLR  FUNCTION
                RTS

* If startup token, get B1. If first time getting 6, increment and store
*      as my address. If second time, increment and store as number of
*      stations. Pass both on to inner Hc11

COMP6           EQU   *
```

```
              CMPA #$06
              BNE  COMP7
              JSR  GETBS
              LDAA DC
              JSR  SENDCHAR
              LDAA BYTE1A
              JSR  SENDCHAR
              LDAA SECOND
              BNE  SECONDTIME
              LDAA #$01
              STAA SECOND
              LDAA BYTE1A
              INCA
              STAA MYADDRESS
              RTS

SECONDTIME    EQU  *
              LDAA BYTE1A
              INCA

*  If number of stations = my address, this station must be station 0

              STAA NOSTATS
              CMPA MYADDRESS
              BNE  RETURN
              LDAA #$00
              STAA MYADDRESS
              STAA SECOND

RETURN        EQU  *
              RTS

*  If D = 7, test if it belongs to me.  If it does, go to send mode

COMP7         EQU  *
              JSR  GETBS
              LDAA BYTE1A
              CMPA MYADDRESS
              BNE  NOTMINE2
              LDAA #$01
              STAA FUNCTION
              BRA  THATSIT

*  If not, go to receive
```

```
NOTMINE2        EQU   *
                CLR   FUNCTION
                LDAA  DC
                JSR   SENDCHAR
                LDAA  BYTE1A
                JSR   SENDCHAR

THATSIT         EQU   *
                RTS

*  If D = 8, test bytes for accuracy.  Send D and both B's on to inner Hc11

COMP8           EQU   *
                BITA  #$04
                BNE   GO12
                CMPA  #$08
                BNE   COMP9
                JSR   GETBS
                JSR   COMPARE
                LDAA  DC
                JSR   SENDCHAR
                LDAA  DC
                CMPA  #$08
                BNE   RETURN
                LDAA  BYTE1A
                JSR   SENDCHAR
                LDAA  BYTE2A
                JSR   SENDCHAR
                RTS

GO12            EQU   *
                JMP   COMP12

* If D = 9, check accuracy then send D, B1 and B2 to inner if okay.
*       If not, only send D

COMP9           EQU   *
                CMPA  #$09
                BNE   COMP10
                JSR   GETBS
                JSR   COMPARE
                LDAA  DC
                JSR   SENDCHAR
                LDAA  DC
                CMPA  #$09
```

```
                BNE   OUTTAHERE
                LDAA  BYTE1A
                JSR   SENDCHAR
                LDAA  BYTE2A
                JSR   SENDCHAR

OUTTAHERE       EQU   *
                RTS

*  If D = 10, check accuracy of B's.  Send D to inner.  If accuracy okay, send both
*       B's to inner

COMP10          EQU   *
                CMPA  #$0A
                BNE   COMP11
                JSR   GETBS
                JSR   COMPARE
                LDAA  DC
                JSR   SENDCHAR
                CMPA  #$0A
                BNE   OUTTAHERE
                LDAA  BYTE1A
                JSR   SENDCHAR
                LDAA  BYTE2A
                JSR   SENDCHAR
                RTS

*  If D = 11, test B's.  Send D.  Send B1 if okay.

COMP11          EQU   *
                JSR   GETBS
                JSR   COMPARE
                LDAA  DC
                JSR   SENDCHAR
                LDAA  DC
                CMPA  #$0B
                BNE   RET
                LDAA  BYTE1A
                JSR   SENDCHAR
                LDAA  #$02
                STAA  FUNCTION

RET             EQU   *
                RTS
```

```
* If D = 12, send D to inner and test B's. If B's okay, send first to inner.

COMP12      EQU  *
            BITA #$02
            BNE  COMP14
            JSR  GETBS
            JSR  COMPARE
            LDAA DC
            JSR  SENDCHAR
            CMPA #$0C
            BNE  RET
            LDAA BYTE1A
            JSR  SENDCHAR
            RTS

* If D = 14, test B's. Okay, send all 3. Otherwise, send only D.
*     Go to skip mode

COMP14      EQU  *
            CMPA #$0E
            BNE  COMP15
            JSR  GETBS
            JSR  COMPARE
            LDAA DC
            JSR  SENDCHAR
            LDAA DC
            CMPA #$0E
            BNE  GONE
            LDAA BYTE1A
            JSR  SENDCHAR
            LDAA BYTE2A
            JSR  SENDCHAR
            LDAA #$02
            STAA FUNCTION

GONE        EQU  *
            RTS

* If D = 15, test B's. okay, send all three, otherwise, send only D

COMP15      EQU  *
            JSR  GETBS
            JSR  COMPARE
            LDAA DC
            JSR  SENDCHAR
```

```
CMPA #$0F
BNE   GONE
LDAA BYTE1A
JSR   SENDCHAR
LDAA BYTE2A
JSR   SENDCHAR
RTS
```

```
DELAY       EQU  *
            PSHA
            LDAA #$10

DELAY2      EQU  *
            DECA
            BNE  DELAY2
            PULA
            RTS

* Does the handshaking and sends characters between inner and outer HC11's

SENDCHAR    EQU  *
            LDAB PORTA
            BITB #$01
            BEQ  SENDCHAR
            LDAB #$40
            STAA PORTCL
            STAB PORTA
            JSR  DELAY
            CLR  PORTA

LOOPY       EQU  *
            LDAA PORTA
            BITA #$01
            BNE  LOOPY
            RTS

* Gets B1 and B2 from the ISDN frame

GETBS       EQU  *
            LDX  TAIL
            LDD  0,X
            LSLD
            LSLD
            STAA BYTE1A
            LSRB
            LSRB
            TBA
            INX
            INX
            LDAB 0,X
            LSRD
            STAB BYTE1B
            INX
```

```
                LDD  0,X
                LSLD
                LSLD
                STAA BYTE2A
                LSRB
                LSRB
                TBA
                INX
                INX
                LDAB 0,X
                LSRD
                LSRD
                LSRD
                STAB BYTE2B
                RTS

*  Compares the first byte of B1 to the second byte of B1 to make sure
*       they are opposites.  Does the same for B2 bytes.  Changes D if
*       there is an error.

COMPARE         EQU  *
                LDAB BYTE1A
                BITB BYTE1B
                BNE  MISTAKE
                LDAB BYTE2A
                BITB BYTE2B
                BNE  MISTAKE
                RTS

MISTAKE         EQU  *
                LDAA #$0C
                ADDA FUNCTION
                STAA DC
                RTS

*  Pulls the 4 bits of D from the ISDN frame

GETD            EQU  *
                LDX  TAIL
                INX
                LDAA 0,X
                ANDA #$10
                LSRA
                STAA DC
                CLRA
```

```
                INX
                INX
                LDAB 0,X
                ANDB #$80
                LSLD
                LSLD
                LSLD
                ORAA DC
                STAA DC
                INX
                LDAA 0,X
                ANDA #$10
                LSRA
                LSRA
                LSRA
                ORAA DC
                STAA DC
                IN    X
                LDAA 0,X
                ANDA #$02
                LSRA
                ORAA DC
                STAA DC
                RTS

* Sends the 6 bytes of ISDN frame to inner HC11

SENDISDN        EQU  *
                LDX  TAIL
                LDAA #$06
                STAA COUNT

SENDISDN1       EQU  *
                LDAA PORTA
                BITA #$01
                BEQ  SENDISDN1
                LDAB 0,X
                INX
                LDAA #$60
                STAA PORTA
                STAB PORTCL
                JSR  DELAY
                LDAA #$20
                STAA PORTA
```

```
WAIT        EQU   *
            LDAA  PORTA
            BITA  #01
            BNE   WAIT
            LDAA  COUNT
            DECA
            STAA  COUNT
            BNE   SENDISDN1
            LDX   #ISDN
            STX   HEAD
            STX   TAIL
            CLR   PORTA
            RTS


*  Variables:

SECOND      RMB   1
MYADDRESS   RMB   1
NOSTATS     RMB   1
COUNT       RMB   1
FUNCTION    RMB   1
DC          RMB   1
BYTE1A      RMB   1
BYTE2A      RMB   1
BYTE1B      RMB   1
BYTE2B      RMB   1

*  Pointers:

HEAD        RMB   2
TAIL        RMB   2

*  Room for ISDN queue

ISDN        RMB   256

*  Room for system stack:

STACK       RMB   1
```