RING-STRUCTURED MICROPROCESSOR NETWORK
FOR MULTIPROCESSOR RESOURCE SHARING

by

DANIEL KELVIN JACKSON

SUBMITTED IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREES OF
BACHELOR OF SCIENCE AND MASTER OF SCIENCE
IN ELECTRICAL ENGINEERING

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

January, 1975

Signature of Author ........................................

Department of Electrical Engineering, January 31, 1975

Certified by ........................................

Thesis Supervisor

Accepted by ........................................

Chairman, Departmental Committee on Graduate Students

MAR 5 1975
RING-STRUCTURED MICROPROCESSOR NETWORK
FOR MULTIPROCESSOR RESOURCE SHARING

by

DANIEL KELVIN JACKSON

Submitted to the Department of Electrical Engineering on January 31, 1975, in partial fulfillment of the requirements for the degrees of Bachelor of Science and Master of Science in Electrical Engineering

ABSTRACT

A ring-structured network of microprocessors, intended to realize resource sharing and high-speed interprocessor communication among a group of minicomputers and associated I/O devices, is described. In the system (under construction in the Digital Systems Laboratory at M.I.T.) each minicomputer and I/O device is interfaced to a network node which communicates with other nodes via wideband dedicated lines. A 'network interface' exists at each node to handle all aspects of message transmission and reception, leaving the microprocessors free to manage system resources.

Data is specially encoded and sent over the network at 7 Megabits/sec. A phase-locked loop is used at each node for clock regeneration. The incoming signal is inspected for encoding violations as a first-order check on data validity. The node that establishes network timing via its crystal oscillator makes use of a simultaneous read/write FIFO buffer so that information can be passed completely around the ring.

Control of the network for purposes of message transmission is passed from node to node. A 'resync' sequence of operations is invoked by any node whenever it is felt that network control may be in jeopardy. 'Resync' removes control from all nodes and causes network activity to cease temporarily. The network-timing node is then chosen, and data reception circuitry at each node is allowed to stabilize. Finally, the network is "restarted" and continues from where it left off. Higher-level logic in the 'network interface' sees to any necessary retransmissions when a message has not been properly received.

Each node contains hardware assemblies to realize a processing system: CPU (an Intel 8080), ROM, RAM and I/O interfaces. A memory controller insures rapid-enough access to data and implements two circular buffers to hold messages before and after transmission.

The system is being debugged and is not yet operational.

Thesis Supervisor: Hoo-min D. Toong
Title: Assistant Professor of Electrical Engineering
ACKNOWLEDGEMENTS

Many people have been involved with various portions of the project and/or the preparation of this thesis, and I am indebted to them all.

The following were instrumental in transforming the schematics into physical form: Keith B. Amundsen, Robert C. Laurenson, Thomas K. Lee, Paul Menig, Robert A. Van der Kloot and especially Russel Kao.

Debugging was (and/or is being) carried out by: Jeffery A. Grossman, Daryl Kinney and Dave Lipschutz (who also helped with the implementation).

J. Elliot B. Moss, in ways too numerous to mention, was generally helpful and encouraging throughout.

With regard to the thesis itself, Ingeborg Meyer helped prepare drawings, and Hannah Allen Abbott took care of the illustrational material. The keyboard artistry of Aina Sils is also greatly appreciated.

Dave Moberly deserves special mention for having put up with an awful lot.

Lastly, I'd like to express my appreciation to Professor Hoo-min D. Toong for permitting me to lead such an interesting and challenging project.
# TABLE OF CONTENTS

Abstract .......................................................... 2
Acknowledgements ........................................... 3
List of Figures ................................................ 7
I. Introduction ................................................ 8
   A. Overview ............................................... 8
      1. Node Construction ................................ 8
      2. Rate of Data Transfer ............................ 9
   B. Network Topology .................................. 11
   C. Message Buffering .................................. 12
II. Network Operation ...................................... 14
   A. Data Transmission Considerations ............. 14
      1. Data Encoding and Clock Recovery .......... 14
      2. Data Recovery and Encoding Error Detection 15
      3. Node Timing and Phase Gap Existence ........ 15
      4. Phase Gap Compensation ....................... 16
   B. Network Control Protocol ......................... 17
      1. Resync Recognition ................................ 18
      2. Choice of Ring Master .......................... 19
      3. Phase-Locked Loop Settling .................... 19
      4. Transmission Control ......................... 21
   C. Message Formatting and Transmission .......... 22
      1. Corruption of Information ..................... 22
      2. Message Control Field ........................ 23
      3. Message Data Field ............................ 23
      4. Message Response Field ....................... 24
      5. Resync Summary ................................ 25
      6. Retransmission .................................. 26
      7. Reception Serialization ....................... 27
      8. Transmission Serialization .................. 27
Table of Contents (cont.)

III. Node Architecture
   A. Hardware Configuration 29
   B. Device Interfacing 29
      1. Programmed I/O 30
      2. Interrupt Operation 30
      3. DMA Operation 31
      4. Lower Memory 32
   C. CPU Board 33
   D. Console 34
      1. Capabilities 35
      2. Realization 35
      3. Implementation 36
   E. Memory Controller 39
      1. Circular Buffer Philosophy 40
      2. Upper Memory Access 41
      3. Pointer Manipulation 42
      4. Implementation 43
   F. Network Interface 46
      1. Message Representation in Memory 47
      2. Control and Status Registers 48
      3. Message Retransmission after an Error 49

IV. Software Overview
   A. Message Protocol 50
      1. Message Types 50
      2. Network Integrity 51
   B. Resource Management 51
      1. Processor Nodes 52
      2. Peripheral Nodes 52
      3. Nonsharable Device Management 53

V. Conclusions and Recommendations
   A. Research Suggestions 56
   B. Project Status 57

Bibliography 58
<table>
<thead>
<tr>
<th>Appendix</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Appendix I:</td>
<td>Data Preprocessor Schematic</td>
<td>78</td>
</tr>
<tr>
<td>Appendix II-A:</td>
<td>Data Preprocessor Exerciser Schematic</td>
<td>86</td>
</tr>
<tr>
<td>&quot; Appendix II-B:</td>
<td>Data Preprocessor Exerciser State Flowchart</td>
<td>105</td>
</tr>
<tr>
<td>Appendix III:</td>
<td>ROM board Schematic</td>
<td>110</td>
</tr>
<tr>
<td>Appendix IV:</td>
<td>RAM Board Schematic</td>
<td>118</td>
</tr>
<tr>
<td>Appendix V-A:</td>
<td>Line Printer Interface Schematic</td>
<td>132</td>
</tr>
<tr>
<td>&quot; Appendix V-B:</td>
<td>Line Printer Interface Registers Description</td>
<td>138</td>
</tr>
<tr>
<td>Appendix VI-A:</td>
<td>CPU Board Schematic</td>
<td>140</td>
</tr>
<tr>
<td>&quot; Appendix VI-B:</td>
<td>I/O Port-Number Assignments and Device Register Descriptions</td>
<td>163</td>
</tr>
<tr>
<td>Appendix VII-A:</td>
<td>Console Schematic</td>
<td>167</td>
</tr>
<tr>
<td>&quot; Appendix VII-B:</td>
<td>Console FSM Flowchart</td>
<td>193</td>
</tr>
<tr>
<td>&quot; Appendix VII-C:</td>
<td>Console Panel Layout</td>
<td>198</td>
</tr>
<tr>
<td>Appendix VIII-A:</td>
<td>Memory Controller Schematic</td>
<td>200</td>
</tr>
<tr>
<td>&quot; Appendix VIII-B:</td>
<td>Memory Controller FSM Flowchart</td>
<td>216</td>
</tr>
<tr>
<td>&quot; Appendix VIII-C:</td>
<td>Circular Buffer Pointer Table</td>
<td>220</td>
</tr>
<tr>
<td>&quot; Appendix VIII-D:</td>
<td>Circular Buffer Flag Operations Summary</td>
<td>223</td>
</tr>
<tr>
<td>Appendix IX:</td>
<td>Integrated Circuit Pin-Number Specifications</td>
<td>226</td>
</tr>
<tr>
<td>No.</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>-----</td>
<td>------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>1.</td>
<td>Network as viewed from a resource.</td>
<td>62</td>
</tr>
<tr>
<td>2.</td>
<td>Possible resource/node configurations.</td>
<td>62</td>
</tr>
<tr>
<td>3.</td>
<td>Node structure.</td>
<td>62</td>
</tr>
<tr>
<td>4.</td>
<td>Synchronous data recovery.</td>
<td>63</td>
</tr>
<tr>
<td>5.</td>
<td>Message format.</td>
<td>64</td>
</tr>
<tr>
<td>6.</td>
<td>Message control field.</td>
<td>64</td>
</tr>
<tr>
<td>7.</td>
<td>Message control field length specification bits.</td>
<td>64</td>
</tr>
<tr>
<td>8.</td>
<td>Message data field.</td>
<td>64</td>
</tr>
<tr>
<td>9.</td>
<td>Message response field.</td>
<td>65</td>
</tr>
<tr>
<td>10.</td>
<td>Message response bits specification.</td>
<td>65</td>
</tr>
<tr>
<td>11.</td>
<td>Example serialization table layout.</td>
<td>65</td>
</tr>
<tr>
<td>12.</td>
<td>Node functional block diagram.</td>
<td>66</td>
</tr>
<tr>
<td>13.</td>
<td>Backplane layout.</td>
<td>66</td>
</tr>
<tr>
<td>14.</td>
<td>Upper backplane signals list.</td>
<td>67</td>
</tr>
<tr>
<td>15.</td>
<td>Lower memory area backplane signals list.</td>
<td>68</td>
</tr>
<tr>
<td>16.</td>
<td>Network interface ID register.</td>
<td>69</td>
</tr>
<tr>
<td>17.</td>
<td>Network interface control register.</td>
<td>69</td>
</tr>
<tr>
<td>18.</td>
<td>Network interface status register.</td>
<td>69</td>
</tr>
<tr>
<td>19.</td>
<td>Network interface interrupt enable register.</td>
<td>70</td>
</tr>
<tr>
<td>20.</td>
<td>Network interface error register.</td>
<td>70</td>
</tr>
<tr>
<td>21.</td>
<td>Data preprocessor.</td>
<td>71</td>
</tr>
<tr>
<td>22.</td>
<td>Data preprocessor exerciser.</td>
<td>71</td>
</tr>
<tr>
<td>23.</td>
<td>ROM board.</td>
<td>72</td>
</tr>
<tr>
<td>24.</td>
<td>RAM board.</td>
<td>72</td>
</tr>
<tr>
<td>25.</td>
<td>Line printer interface.</td>
<td>73</td>
</tr>
<tr>
<td>26.</td>
<td>CPU board.</td>
<td>73</td>
</tr>
<tr>
<td>27.</td>
<td>Console.</td>
<td>74</td>
</tr>
<tr>
<td>28.</td>
<td>Console front panel.</td>
<td>74</td>
</tr>
<tr>
<td>29.</td>
<td>Memory controller.</td>
<td>75</td>
</tr>
</tbody>
</table>
I. INTRODUCTION

This document describes a system [under construction in the Digital Systems Laboratory (DSL) of the Electrical Engineering Department of the Massachusetts Institute of Technology] to interconnect a half-dozen minicomputers (of various manufacture) and associated I/O devices in such a fashion that resource sharing and high speed interprocessor communication may be achieved.

A. Overview

1. Node Construction

A resource (defined as an I/O device or a processor with associated I/O devices) is able to communicate with other resources via an interconnection network made up of nodes and links (Fig. 1). The links are the means whereby information travels from node to node, while the nodes act as access points through which data passes from the resource out onto the network (or back the other way). For convenience, nodes that interface processors to the network may be termed 'processor nodes', while nodes associated with I/O devices alone may be described as 'peripheral nodes' (all nodes are identical hardware-wise; the names are used merely to differentiate the nodes' functions).

It is helpful to distinguish between processor nodes on the basis of whether or not an operating system (OS) is supported by the hardware (see Fig. 2). If an OS exists, management of I/O devices interfaced to the CPU can be taken care of by the OS itself. If the processor cannot support an operating system, any associated I/O devices are not accessible by the rest of the network since only one process (typically user-program
related) can run on the CPU. In this case, as well as with peripheral-type nodes, it is clear that management of the resource cannot be carried out by the resource itself. Since in the DSL most computers are not able to support operating systems, it was decided to incorporate into each node the hardware necessary to realize execution of software. The job of these processing units at each node is to manage the node's resources and/or, together with the network-communication hardware, see to the interchange of messages with other nodes.

Figure 3 indicates in general terms the hardware makeup of each node, the heart of which is a microprocessor (integrated processing unit). The resource (processor or I/O device) makes connection to the node hardware through suitable interfacing logic, and the node itself connects to the rest of the network through a 'network interface'.

2. Rate of Data Transfer

Once a large capacity disc is interfaced to the network, a secondary storage medium will become available for the first time to many of the DSL minicomputers. It is expected that heavy use will be made of the network in supporting the information flow to and from such a disc. Hence, it is desired that the system achieve a high rate of data exchange from node to node.

In addition to the sharing of resources, the existence of a processor-interconnection network will allow important experiments related to distributed computing and distributed operating systems (consult Refs. 1, 2, 3, 4, 5) to be carried out. Many of the more interesting problems involving distributed multiprocessing where large volumes of data are communicated between the processors over the network links
would prove to be I/O bound if the communication pathways were to operate at commonly-encountered asynchronous I/O-terminal rates. System response time and throughput would be low then and the types of problems that could realistically be programmed would be restricted. For this reason then, too, a major design goal was to have the system be capable of moving data quite quickly.

Another network⁶ achieves only a one kilobyte (8000 bits) per second transfer rate (approximately) because there the node microprocessor is intimately involved in all aspects of information transfer over the network. The DSL system hopes to realize much higher speed communication, on the order of one byte per microsecond, by utilizing direct memory access (DMA) techniques. Such speed, even when viewed solely in the context of having the system provide no services other than the sharing of resources, is hardly excessive when the relatively low rate of instruction execution by the node microprocessor is considered. On the other hand, the amount of time needed by the node software to attend to message-level network protocol will not be so large that the specialized internode communication facility will be wasted.

The desire for high-speed operation directly affects the mechanics of the data transmission process as well as the architecture of the node itself, as will be seen; the result is a hardware/software tradeoff wherein all aspects of message transmission and reception are taken care of by the hardware, with message formulation and interpretation carried out by software.
B. Network Topology

The manner of interconnecting the nodes of the network should result in a system that is simple to maintain, is readily expandable and contractible (in terms of the number of nodes connected to the network), and is not inherently subject to catastrophic failure. Since all minicomputers are treated alike with regard to priority of access to shared devices, a topologically symmetrical network was favored.

Three symmetrical interconnection schemes spring to mind. One, for an N node network, has each node connected to the other N-1 nodes, via N-1 individual links. This method of node interconnection (or a similar one, with somewhat fewer links) does not lend itself to ready expansion, is quite expensive, and its high reliability is difficult to efficiently take advantage of in the DSL (where all computers are located in the same general physical area) to justify its expense.

The second configuration, the "star", has one central node, with all other nodes connected to that one via outward-radiating links. Here, data is routed by the central member, which acts as a multiplexer. Expansion of this type of network is not readily accomplished, but even more importantly, the reliability of the whole network hinges on the center node. Should it fail, all communication would be affected and the network would remain down until repairs could be implemented or a spare multiplexer brought on-line.

The topology of the DSL network is that of a third kind of interconnection, a "ring", where each node is connected to its two neighbors such that a circle is formed. Expansion and contraction are readily
achieved by breaking the ring and adding a new node (or removing an already existing one). Most importantly, no one node is vital to the operation of system. Since all units are capable of performing identically, if a node fails, the network can be made immediately operational once again by removing the defective node. While the central node of the star network performs the switching function in that configuration, here each node is capable of deciding, on the basis of control information passed along the ring with the transmitted data, whether the message is intended for it or not.

C. Message Buffering

One of two different methods may be used by the network nodes to pass information around the ring. The Distributed Computing System is an example of what may be described as a 'buffered' ring, where each node receives and buffers all bits of each incoming message before sending them back out again. Since all nodes are constantly sending and receiving information, there are N (for N nodes) message "slots" extant at any time on the network, and therefore up to N messages in transit.

In a 'nonbuffered' ring (see Ref. 7, for example), only one node is transmitting information out over the network at any one time. The message speeds around the ring with minimal delay at each node (typically one bit). There is usually at most one message on the ring at a time, but there may be more, depending upon the total delay experienced by the data as it travels around the ring.

With either 'buffered' or 'nonbuffered' versions, some means must exist of unambiguously keeping the node synchronized with the other nodes.
That is, if the Kth bit received in heralds the start of a new message, then all nodes should so interpret that bit (and not think it perhaps to be data). No one node is ever "control" of a 'buffered' ring, since all nodes are sending constantly. A 'non-buffered' ring must have provision for transfer of control from one node to the next.

With the 'buffered' approach, the message size is fixed (small messages must be padded and large ones broken up), and each node must have a buffer constantly available in which to hold the incoming message.

In the 'nonbuffered' case, the size of the transmitted message is variable and only a one bit "buffer" is needed at each node to "hold" the passing message. The DSL ring network is of the 'nonbuffered' variety as a result of consideration of all the above mentioned factors.

The remainder of this thesis gives a more detailed account of all aspects of the DSL network operation (Chapter II), including a discussion of the hardware assemblies that together form a working node (Chapter III). Although implementation of software for the system was not the author's responsibility, no discussion of hardware can be divorced from consideration of the software that will make use of it; Chapter IV, then gives a very brief overview of what that software (when written) will be called upon to do. Finally, suggestions for future research involving the DSL ring as well as a system progress report are given in Chapter V.
II. NETWORK OPERATION

A. Data Transmission Considerations

1. Data Encoding and Clock Recovery

Synchronous transmission of data is employed so that long messages may be reliably sent at high speeds. Since it was desired to send a single transmission line around the ring from node to node, the data is encoded in a special format so that clock information as well as the data may be decoded from the composite signal. As illustrated in Fig. 4, a di-phase encoding scheme is used, wherein a transition always occurs on the line at the end of the data period, with a transition in the middle of the period only if the data bit is a "1". The clock may be recovered by processing the signal received in from over the link (realized by a shielded twisted pair of wires) by firing a monostable at the end-of-period transitions, with the timing adjusted to 3/4 of the data period. As soon as a "0" is received in, the circuitry automatically synchronizes itself with the proper transitions. A phase-locked loop (PLL) is synchronized to the monostable's output so that a relatively stable timing source may be had, one not affected by phase jitter, noise or dropouts that could occur on the line. The center frequency of the PLL is adjusted to correspond to the frequency of the incoming data, in this case 7 MHz. (This frequency was dictated by a number of factors, including network interface finite-state machine constraints as well as the access time of readily available RAM for use in conjunction with the node microprocessor.) Thus, the fundamental bit-time is 143 nsec, with signals being asserted on the line for as little as 71.5 nsec.
2. **Data Recovery and Encoding Error Detection**

Examination of the encoding format reveals the fact that the information is present in total redundancy. This permits a close check to be performed on the validity of the incoming data, if the signal is sampled at twice the fundamental frequency (see Fig. 4). Thus, the PLL is set up to provide not only a 7 MHz signal but also a double frequency (14 MHz) clock. Now, if the encoded signal is sampled by the faster clock on each rising edge and the samples processed by a small finite-state machine (FSM), data and 'encoding error' signals (both synchronous relative to the 7 MHz PLL clock) may be derived. The data signal is a delayed version of the exclusive-Or (XOR) of two samples taken inside a 143 nsec period that coincides with the beginning and end of a data period, while the error signal is a delayed version of the inversion of the XOR of two samples taken on either side of a data period boundary. The error is asserted if the di-phase encoding format was found to be violated at what corresponded to the end of the present bit-time; this is the ring's first-order (most fundamental) error detection mechanism.

3. **Node Timing and Phase Gap Existence**

The PLL 7 MHz clock is used at most nodes to provide timing to perform all FSM operations of the network interface, including the encoding of the node's data for transmission downstream to the next node (the encoding may easily be done synchronously by making use of the 14 MHz clock additionally). Therefore, the node derives its timing via its own PLL from the signals sent by the preceding node. At one node a crystal oscillator is used instead to time the FSM operations and to therefore provide a timebase for the rest of the network. Although
it would be possible to dedicate a special-purpose node to take care of network timing (as in Ref. 7), each node has the capability (via a procedure described later) of becoming the timing node [the 'ring master' (RM)], so that all nodes are indeed identical.

It is easily seen that something of a problem arises when the RM attempts to process the signal coming back into it from around the ring. The problem is that the delay experienced by the signal in traversing the loop has an arbitrary value compared to the basic data bit period due to the propagation delay down the links and through the circuitry of each node. This means that the returning signal has an unknown phase relative to the signal originated by the RM node, and thus cannot be readily utilized by the crystal-operated FSM logic of that node. The situation is even worse because the phase difference (or 'phase gap') will be a (slowly varying) function of temperature; as the propagation delays of the various links and nodes change, the size of the phase gap will change, too. If it could be stated that the RM never has to pass information from its input through to its output for retransmission, then things would be fine. However, in the general case the RM node will not be the node that is originating a message for network transmission, and must, like the other non-source nodes, send the information back out again. Therefore, the data signal must be made to bridge the phase gap somehow. Also, because of the way the encoding error is used by the network interface, that signal too must be similarly handled.

4. **Phase Gap Compensation**

A first-in-first-out (FIFO) buffer capable of being simultaneously
written into and read from is implemented to overcome the phase gap, one for each of the two signals of interest. The buffers are one bit wide by eight words long and are set up to be initially half full. The incoming data signal is decoded by the PLL-related circuitry of the RM node (just as with all other nodes), and each data (and encoding error) bit is written into the proper buffer. Information is read out by the RM node's crystal-operated FSM logic on "the other side" of the phase gap. Even if the size of the gap changes after operation of the network is begun, the only thing that will happen is that the buffer will become fuller (shorter loop delay) or more empty (longer delay). Data transmission becomes impossible if the limits of the buffer are reached; the action then taken is discussed in the section on control protocol.

It should be restated for emphasis that any node may become RM (and that therefore all nodes have crystal oscillators and FIFO buffers), but that only one node is RM at any one time. The design of that part of the network interface logic that decodes the high frequency data stream, implements the PLL and contains the FIFO phase gap circuitry (called the 'data preprocessor'), is described in Appendix I.

B. **Network Control Protocol**

It is the job of the 'resync' protocol sequence of operations, carried out by the individual nodes in concert, to bring the system to a state where message transmission around the ring may be carried out. A resync is initiated when a node wants to join the network, but is also employed when any one of a number of error conditions is detected by any node, as will be discussed later. For now, consider a node that has just been powered up and wants to function as part of the network over which
other nodes are already sending messages; this brings up the question of how information is relayed down the line by nodes that are not powered up. This is taken care of by having a small portion of the network interface logic (three integrated circuits) kept turned on whenever use is to be made of the ring. Thus, the signal is received and sent back out again with no processing having been performed by the node. Some time after a node's main power has been applied, however, the ring must effectively be "broken" to admit the new member. The resulting corruption of data is detected by the other nodes as a malfunction (as discussed later), and a resync is initiated by (at least) one of them. The end result of the resync process, we shall see, is that the network will again be able to function properly and pick up from where it left off.

1. **Resync Recognition**

While a complete list of all resync-causing conditions will be given later, suffice it to keep in mind that a resync is initiated by a node detecting an error situation that directly relates to its (or some other node's) ability to keep track of what has been happening on the network. If there is some chance that control of the network is in jeopardy, a resync is called for. In order to communicate the recognition of the resync condition to other nodes, the node will hold its output (di-phase-encoded) signal constant (no transitions). This will be interpreted as an encoding error by the next node, which will cause it to go into resync. Soon, all nodes will have been notified of the resync condition and will hold their outputs clamped for a time period 'T1', about one millisecond. This length of time insures that, on that scale, all nodes enter resync at approximately the same time.
2. **Choice of Ring Master**

The second phase of the resync, 'T2n', is used to determine whether any other node has taken it upon itself to become ring master and "bring up" the network. The value of T2n is different for each node (for example, 2 msec, 3 msec, 4.5 msec, etc.); this insures that only one node is chosen RM. If by the end of the T2n period a node has not detected a signal coming from the preceding node, it declares itself to be RM and commences transmission of a string of "0"'s (which establish proper data preprocessor synchronization). On the other hand, if a signal is detected during T2n, the node does not become RM but starts sending out "0"'s anyway. Therefore, all nodes will soon be transmitting "0"'s to their neighbors. The RM node meanwhile listens for 'T7' to see that in fact some signal has come back around to it; if not, a blockage (perhaps temporary) exists, and another resync is begun to try again to activate the network (T7 is about 1.5 msec, somewhat longer than T1). It is clear that under usual circumstances the node with the shortest T2n period will always become RM; however, all nodes must have the capability of becoming RM in the case no lower T2n-valued nodes happen to be in the network.

3. **Phase-Locked Loop Settling**

As part of the first phase of doing a resync, the data preprocessor is directed to feed the crystal oscillator's output to the PLL; this prevents the PLL's frequency from drifting off center when no signal is received from the node upstream. Once the second part of the resync is over, the third is begun, during which the PLL is again allowed to track the incoming signal. The length of this phase is controlled by a
monostable whose period ('T5') is much longer than the PLL's natural time constant (which is on the order of milliseconds). While the PLL locking process is under way, the output frequency will vary in a damped oscillatory manner; this will be true of the PLL in the first node downstream from the RM, for example. The node past that one will attempt to lock onto the signal it is seeing, with the resultant PLL frequency describable as a self-convolved damped sinusoid. Thus, each node will take longer to achieve lock than the one before it, with the RM PLL the last to finish. Since all nodes are transmitting zeros, the T5 monostable at any particular node is caused to be retriggered to extend the length of this phase of the resync operation if its data preprocessor decodes an incoming bit as a "1", or detects an encoding error. This means that the PLL has not yet achieved lock as the incoming data has not been interpreted correctly. The resync operation is complete when nothing but "0"'s have been received in for the last T5 (23 milli-) seconds. Each node that is not the RM then stops originating its "0"'s and switches over to the nominal idle mode of just retransmitting the data that was received in and decoded by the data preprocessor.

While proper reception of incoming data at most nodes may be assured once the PLL frequency is approximately correct, at the RM node account must also be taken of the FIFO buffers; the phase gap compensation won't work if the buffers become too empty or full as a result of the RM PLL going through the locking process. Thus, the T5 monostable of the RM node is caused to be retriggered not only if "0"'s are not being received in, but also if a phase gap buffer underflow or overflow is
detected. Once the RM PLL has settled down sufficiently so that communication may take place around the whole ring, the RM node outputs one "1" and then an indefinite number of "0"'s until the "1" has come back around again. Finally, that node, too, then switches into the nominal idle mode.

4. **Transmission Control**

The pattern of a "10" (with a string of trailing "0"'s) cycling repeatedly around the ring is what is observed on the network in the steady state when no message origination is taking place. The initial "1" implies existence of control information as contained in the next bit; the "0" indicates that any node may begin transmission of data over the network. This involves changing the "0" to a "1" and then appending a message. (Such bit-permutation is easy to accomplish of there is a one bit delay at the node; in the DSL ring the data preprocessor circuitry inserts an additional three bit delay, bringing the total to four.) Each node recognizes the "11" control header to mean that message information follows immediately. The message sent out is relayed by each node in turn until it returns to the source node [called the 'ring lor_' or 'ring leader' (RL)]. Appended to the message is a "1" followed by an indefinite number of "0"'s; the source node goes back into idle mode when the "1" is received back in again. Potential control of the network is thereby passed on to other nodes via the trailing "10" control header. Each node fires a 'T4' monostable (whose period is approximately 15 microseconds) after receiving a message or seeing a "10" control header. Timeout of the monostable will automatically cause a resync if no message is being received in at the time; this protects against loss of the control header.
A circuit was designed (the 'data preprocessor exerciser' -- see Appendix II) to test all aspects of network control protocol and to debug the data preprocessor.

C. Message Formatting and Transmission

1. Corruption of Information

Every message is composed of three information fields: the 'data' field, a preceding 'control' field and lastly a 'response' field (see Fig. 5). Before describing the makeup of each of the fields, it should be pointed out that an encoding error, if not received during the data field portion, will cause an immediate resync. The data portion of the message is handled differently because it is potentially more than two orders of magnitude larger than the non-data areas, and therefore much more likely to become corrupted during transmission. Further, such corruption, if limited to the data field, does not influence a node's ability to know what is transpiring on the network, whereas a bad control or response field could do just that. So, while a resync is not initiated right away, some means must nevertheless exist of informing downstream nodes of bad incoming data; this is accomplished by purposefully mis-encoding the information sent out by the node in this circumstance. (This implies that, as mentioned earlier, the encoding error signal must be able to jump the phase gap along with the data.) Differentiation between occasional data field encoding errors and a full-fledged resync is accomplished by having the data preprocessor return a signal that is asserted when no transitions have been observed on the incoming line for ten bit-times or so; existence of this 'extended signal loss' indication unconditionally causes a resync. (It is possible that the data pre-
processor clock recovery circuitry can become improperly synchronized to the data stream if signal corruption takes place during a stretch of "1"'s in the data field portion of the message; this would not influence recovery of data, which would be recognized as bad, but would cause the PLL to go through the oscillatory locking process. The complexity of the network interface circuitry implies that a malfunction of the logic would result if the clocking frequency were allowed to move greatly upward from its nominal value of 7 MHz; thus, a 'CFU' signal from the data preprocessor causes a resync if the node's PLL period becomes too small.

2. **Message Control Field**

Figure 6 indicates the function of the five control field bytes. Each node is assigned a unique ('ID') number and is thereby communicated with; a destination value of all "0"'s indicates a message meant for all nodes (an 'all-points bulletin'). The number of bytes of information in the upcoming data field is specified in the 'length-low' byte and in the four lower bits of the 'length-high' byte; the effective value may be calculated by preceding those twelve bits with a "1" and taking the two's complement of the result. The other bits in that byte perform miscellaneous control functions; 'F1' and 'F0' are ignored by the hardware but are delivered with the message for interpretation by the software (see Chapter IV), while 'U' and 'S' relate to message serialization (to be discussed shortly). The last part of the control field is the 'check' byte; this is the inverted checksum of the previous four bytes.

3. **Message Data Field**

The second portion of the message contains the actual data to be sent, which may vary in length from 1 to 4096 bytes (see Fig. ?). A one
byte inverted checksum (across the data bytes only) is appended.

4. **Message Response Field**

The remainder of the message is devoted to allowing the destination node(s) to send a one byte acknowledgement back to the source to indicate whether any problems were encountered trying to copy the information. A one byte inverted checksum (of just the response byte) exists for similarity with the other two message fields. The source node uses the information returned in the response byte to help determine whether a retransmission is called for. It should be noted that only those nodes addressed by the destination control byte are allowed to modify the response byte as it goes by.

The 'destination matched' bit is asserted if the node recognizes that the message is intended for it; a value of "0" read by the source node would imply non-existence of the intended destination node. The 'destination already matched' bit becomes set if the node is addressed but bit #6 (see Fig. 10) was already set. The bit normally becomes set as a result of an all-points bulletin, but otherwise indicates an error.

The 'unable to process message' bit is turned on by the destination node if the microprocessor software has directed the hardware not to deposit any information in memory (see Chapter III). Bit #3 is turned on if serialization table information (as discussed later) directs that the message be ignored as a retransmission; this may conditionally be interpreted as an error by the source node. The 'bad data' bit is asserted if an encoding error was detected during the data field or if the data field check byte was found to be incorrect. Lastly, if there was some problem at the destination node with regard to being able to dispose of the data
(a slowly responsive memory or a buffer overflow), the 'destination memory error' bit is set by the destination node. See Chapter III for a discussion of software handling of errors reported by the network interface.

5. Resync Summary

It should be recognized that "soft" errors that have nothing to do with whether the network is being controlled properly are the sort reported in the response field. A resync is instituted, on the other hand, in more serious cases, as now outlined:

1) The phase gap buffers' limits have been exceeded; data transmission cannot take place around the ring. As part of the resync process the RM node re-establishes the nominal four bit gap compensation, and network operation may continue on.

2) The data preprocessor reports that the PLL frequency has risen too high. Detection of this is necessary only because the speed of data transmission is so high; if it were less, PLL frequency variations would be totally unimportant.

3) An encoding error or a bad check byte is detected during the control or response portions of a message.

4) The data preprocessor has asserted the 'extended signal loss' line, indicating that the upstream node has initiated a resync.

5) No control header has been seen in the last 'T4' seconds. A new header is regenerated by the RM at the end of the resync.

6) The source field of the message coming back into the source node does not have that node's ID number; evidently, some other node has decided to become RL also. Control is removed from all nodes via the
resync process, and network operations are restarted properly.

7) The source node experiences a memory error (memory response time too slow or a buffer underflow); the resync informs all destination nodes that this message is no good and should be ignored. Note that this situation could have instead been handled by purposeful generation of an incorrect data check byte; however, here (as elsewhere) the intent is never to rely on the second-order error detection mechanisms alone to reveal the existence of improper data conditions.

6. **Retransmission**

When there is a problem in message delivery [whether a "hard" error related to data corruption (resulting in a resync) or a "soft" error having rather to do with a node's ability to process a message intended for it (reported in the response field)], the source node recognizes that the transmission, for whatever reason, was unsuccessful and must be attempted again. To prevent hogging of the network, a node may become RL for transmission of exactly one message; control is then always passed on so that other nodes may use the ring. At some time in the future the node will again be in position to become RL and will want to send the message out a second time. Informing the destination node(s) that the message is a retransmission will, of course, not enable them to determine what to do with it. Instead, a one bit 'serialization' value (see Ref. 1) is transmitted in the control field to permit determination of whether the message has been seen and received correctly before; if so, it is ignored this time, but if not, an attempt is made to copy the message.

Each node maintains a 'serialization table' that keeps track of
the current value of the serialization bits to be used in communicating with other nodes. The table has three entries: an 'output' entry, for origination of messages, an 'input' entry, for incoming messages intended exclusively for this node, and an 'all' entry for incoming messages that were destined for all nodes (destination control byte of "0"). All table entries are one bit wide by at least N nodes long and are reset to "0" before the node inserts itself into the ring. To permit receipt by a newly-entered node of messages from those other nodes already operating on the system, two additional entries (the 'input init' and 'all init') exist, also initially reset.

7. Reception Serialization

Messages are normally accepted by a node only if the 'S' bit of the 'length-high' byte of the control field is not the same as the corresponding 'input' serialization table entry (or 'all' entry, if the destination control byte was "0"); if a match occurs, the message has been seen before and is now ignored. If the node has just entered the network, however, it has no knowledge of what value the serialization bit of any incoming message will have; thus, if the 'input init' (or 'all init') table entry is "0", the message is unconditionally accepted, the 'init' entry set to "1", and the 'input' (or 'all') entry properly updated. Completely successful receipt of a message will result in the entry of interest being complemented so that following messages will be properly processed.

8. Transmission Serialization

When a node originates a message, the serialization bit to be transmitted is retrieved from the 'output' table (which includes an additional entry for the case of all-point bulletins). However, if the node had been
active on the network and then was powered off and on again, the other nodes will have what are effectively indeterminate values in their serialization tables due to previous transmissions. Thus, the 'U' bit (serialization 'update') of the control field's 'length-high' byte may be employed by a node newly entering the ring to inform all other nodes of its arrival (perhaps by issuing a special all-points bulletin for this purpose). This bit causes the proper locations of the four incoming-message-related serialization table entries to be reset to "0". A message with the 'U' bit asserted is always received in by the nodes, regardless of the prior values of the table entries.

The successful transmission of a message by a node results in the proper 'output' table entry being complemented. A message that the source thinks might not have been received properly by the destination node(s) will in most cases be retransmitted as soon as the node can become RL again (the exceptions, 'terminal errors', will be noted in Chapter III). Retransmission will continue until either the message is successfully delivered, a terminal error is detected, or four transmission attempts have been made. The node would not try to become RL after that, and the microprocessor software would be informed of the problem. Possible courses of action to be taken at that point are discussed in Chapter III in the section on the Network interface.
III. NODE ARCHITECTURE

A. Hardware Configuration

Each node is composed of a collection of hardware assemblies whose job is 1) to handle all aspects of message reception and transmission over the network, and 2) to provide an environment in which software execution can take place. The 'network interface' carries out network-related operations (as described in Chapter II) and can be thought of as the means whereby connection is made to the rest of the network. The remainder of the hardware (see Fig. 3) realizes a working processing system and provides interfacing capability for connection to I/O devices.

Because of their ready availability, Intel 8080 microprocessors were chosen for use. Each node has a 'CPU board' which contains the microprocessor along with timing and bus-driving and -receiving logic. Connected to the microprocessor's address and data buses (as shown in Fig. 12) are I/O interfaces and at least one 'ROM board' (see Appendix III). ROM (actually erasable programmable-ROM) provides program storage capability at the node. Temporary storage of data is accomplished by using at least one 'RAM board' (Appendix IV). While the usual practice is to have all memory interfaced to and under direct control of the 8080 (consult Ref. 8), faster direct memory access (DMA) operation than that would have provided was needed to support the 7 MHz network transmission rate. This problem was circumvented by splitting the address space of 65K bytes in half; the lower portion (the 'lower memory'), under direct control of the microprocessor, is meant to hold ROM, while the 'upper memory', accessed through a 'memory controller', would provide
RAM for data buffering. Note, however, that the hardware has been implemented such that any mixture of ROM and RAM may be used in the upper and lower memory area; this facilitates program development and debugging. To help along these lines, an "optional" console exists which may be connected in with the rest of the hardware when it is desired to monitor program execution.

In addition to controlling access to the upper memory, the memory controller implements two circular buffers [the 'input circular buffer' (ICB) and the 'output circular buffer' (OCB)] to hold messages before network transmission and after reception by the network interface.

B. Device Interfacing

All node logic assemblies plug into a backplane via 36-pin double-sided connector-edges. Full-size boards have four connector-edges, while half-size ones make connection through only two. Most units have access to the backplane signals carried on the upper two connectors at each board-position (see Figs. 13 and 14) and pass special-purpose signals in and out on the lower two edges. Memory boards implementing the upper memory as well as small I/O interfaces are only half-size and therefore make connection to the bussed backplane signals alone. The lower memory area is a section of the lower backplane across which lower-memory-related signals have been bussed (see Fig. 15); RAM and/or ROM boards comprising the lower memory would be plugged in there.

1. Programmed I/O

The lower eight bits of the microprocessor's address lines ('A7'-'A0') are sent out on the upper backplane (see Fig. 14) to relay I/O
port-number information to the I/O interfaces. The data bus ('D7'- 'D0') is a tri-state bus; the drivers on the CPU board are normally enabled to present data out on the bus, but become disabled when it is desired to read information in from the interfaces (or from memory). Information is read from the bus and latched by a device when its port number appears on the address lines and the 'OUT•WR' signal becomes asserted. Conversely, the microprocessor will be looking for information from the device whose port number is on 'A7'-A0' when 'INP•DBIN' exists. An example circuit (the 'line printer interface'), able to perform basic transfer of data into and out of the CPU, is given in Appendix V-A.

2. **Interrupt Operation**

A serial priority-line interrupt scheme has been implemented to allow I/O devices to interrupt the processor. Two signals, 'IRPIL' and 'IRPOL' are passed from device to device; the physical placement of the interfaces along the backplane determines what priority a device will have. Once it wants to interrupt, the device stops passing priority down to any following devices (refer again to Appendix V-A). It must make sure, however, that it in fact has priority coming into it before asserting the 'IRQL' line; this informs the CPU that an interrupt is desired. When the microprocessor grants the request (by asserting 'IRGL'), the device places interrupt vector information on three of the data lines ('D5','D4' and 'D3') to indicate which servicing routine is to be activated. The device continues to prohibit priority from being passed on until the servicing routine attends to the situation that caused the interrupt (in the case of the line printer, this means either
loading the next character or turning off the interrupt-enable control bit).

3. **DMA Operation**

   DMA can only be done in conjunction with the upper memory, under control of the memory controller. The microprocessor and the network interface, both of which must be able to access that memory area, communicate with the memory controller using special signals not bussed in the upper backplane. DMA I/O devices, however, do communicate via bussed backplane signals, as follows. The 'PERIPHERAL ACCESS REQUEST' line is asserted first by the device. 'PERIPHERAL BUFFER SPECIFICATION' is asserted if a read from the 'input circular buffer' is desired; otherwise, a write into the 'output circular buffer' will be done. The 'PERIPHERAL FSM READY' signal from the memory controller, normally asserted, becomes unasserted. When the controller is able to process the request, the 'PERIPHERAL REQUEST ACKNOWLEDGE' line becomes asserted; this indicates that the device, if writing into the memory, may enable its tri-state buffers and thereby control the 'UMD15'-'UMDO' lines. (The upper memory data bus is 16 bits wide so that a high rate of information transfer is realized.) When the memory controller finishes processing the request, the 'PERIPHERAL FSM READY' line becomes asserted once again.

   Peripheral devices may only do DMA to the circular buffer areas of the upper memory; information about exactly what location the data is going to (or coming from) is already known by the memory controller, as explained in Sect.III.E.2. All upper backplane signals not mentioned so far (with the exception of 'INITL', which is simply the system initiali-
zation signal, asserted at power-up) will be discussed later in connection with the memory controller.

4. **Lower Memory**

The signals in the lower memory area (Fig. 15) originate on the CPU board and control those memory boards being used to implement the lower memory. To provide full compatibility between the upper and lower areas of memory, the lower memory data path is as wide as that of the upper memory, 16 bits. The lower eight bits, 'LMD7'-'LMD0', are electrically the same as (identically equivalent to) the data lines found on the upper backplane, 'D7'-'D0'. Additional lines ('LMD15'-'LMD8') provide the extra eight bits. The 15-bit lower memory address is specified in 'LMA14'-'LMA0', of which 'LMA7'-'LMA0' are identical with 'A7'-'A0' on the upper connectors.

The CPU board has logic to generate the signals needed to control the lower memory. 'ACCESS LMEM' is asserted when an information transfer is desired; 'LMEM BUSY' is sent back by the addressed memory board to indicate the transfer is under way. 'LMEM WRITE' differentiates between a read and a write, while 'LMEM DATA ENABLE' is asserted when the data lines may be driven by the memory board. 'LMEM WORD ACCESS' provides compatibility with the upper memory area (as discussed in the section on the memory controller) and is always held unasserted; this indicates that data should be written (if a write is to be done) into the particular byte specified by 'LMA0' (rather than into the 16-bit word specified by just 'LMA14'-'LMA1').
C. CPU Board

The 8080 (reference may be made to Appendix VI-A to better follow this discussion) is clocked at approximately 2 MHz so that the software will run as quickly as possible. All data buses ('LMD15'- 'LMD8', 'D7'- 'D0' and 'UMD15'- 'UMD0') are received and multiplexed down to a single group of eight lines ('MD17'- 'MDI0') for presentation to the microprocessor; these lines are also buffered and sent to the console for display there. Further multiplexing is performed to allow selection of either the information coming in from memory or I/O devices, or of special data ('CD7Z'- 'CDOZ') the console may want the CPU to see instead (operation of the console is explained in the next section). The resultant data (with provision made for creation of the "RST" instruction during interrupt acknowledging) is fed to the 8080 via a set of tri-state buffers (enabled only when the CPU is accepting data in).

Data coming from the microprocessor is received and buffered by another set of tri-state gates (which the console can disable if it wants to fake the data supposedly originated by the 8080) and eventually gets put out onto the data buses.

The address lines are simply received, buffered and sent out onto the backplane.

Some 'standard software options' (see Appendix VI-B) are associated with the CPU; these are I/O devices that are physically located on the same board as the microprocessor. For example, circuitry exists which causes the 'INITL' backplane signal to become asserted (for approximately 30 milliseconds) when an "OUT" instruction to I/O device #1 is executed. Reading from I/O port #1 returns information that may be useful
in conjunction with servicing interrupts, and includes the interrupting
device's 'subvector specification'. This is simply a record of the
'D6', 'D2', 'D1' and 'D0' lines of the data bus (as asserted by the
device interface during the interrupt acknowledgement sequence) to
indicate exactly which device caused the interrupt when a vector ad-
dress is shared among several interfaces.

A 'programmable stack limit' mechanism (devices #2 and #3) detects
a stack overflow and can inform the program of the address violation.
A 'switch and light panel' is accessed as devices #4 and #5 and permits
exchanges of limited amounts of information between the computer and the
user.

D. **Console**

1. **Capabilities**

   The console with associated front panel (Appendix VII-C):

   1) allows the programmer to examine and change memory locations
      and I/O device registers;

   2) permits instruction execution to be stepped through (one cycle
      or one whole instruction at a time);

   3) displays the CPU's address bus during program execution, and
      memory or I/O port address information during examines and deposits;

   4) displays the CPU's data bus during program execution;

   5) gives status information about the current CPU operation;

   6) monitors and can inhibit program interrupts;

   7) controls a hardware breakpoint facility;

   8) can start (or continue) program execution at any specified
      location;
9) allows the CPU's accumulator register to be examined or changed.

The address to be used in conjunction with an examine or deposit is specified by setting the value in the 16 switches and pressing the 'LOAD ADDRESS' button. Successive examines or deposits of memory locations (not I/O device registers) cause the address to be automatically incremented.

The breakpoint facility is quite extensive; it can be set up to give control to the console when the CPU's address bus has any desired relation (any combination of "greater than", "equal to" and "less than") to the 16 panel switches, and when the CPU is reading or writing a memory location or an I/O device register.

Other miscellaneous capabilities include generation of the 'INITL' backplane signal and the ability to reset ('restart') the CPU.

2. Realization

It was seen that the CPU must deal with several different data buses to handle data flow to and from the upper and lower memories and the I/O devices. Thus, it makes sense to have the console, if possible, not duplicate the effort but try to use the CPU logic that already exists when it is desired to do examines or deposits. Furthermore, because the 8080 microprocessor was not designed to allow direct access to its internal registers, some means has got to be found to implement the loading of arbitrary values into the program counter (PC) if item #8 of the earlier list is to be realized. Therefore, it was decided to have the console logic "force" execution of certain instructions by the CPU to carry out the desired operations. For example, the PC may easily be
loaded by causing a "JMP" instruction to be executed.

To take care of examining and changing memory, the "LDA" and "STA" instructions are used; for I/O registers, "IN" and "OUT" are done. These all work fine except that the accumulator (ACC) is involved in all four cases. The "STA" and "OUT" instructions, for example, send the ACC value out on the data lines so that it will be written into the memory location (or I/O device) addressed. The console must therefore prevent the value on the microprocessor's data lines from being sent out over the backplane data buses, and cause its own data (here, the eight-bit value that is to be deposited) to be sent out instead.

The "LDA" and "IN" instructions take the value on the microprocessor's data lines and load it into the ACC; all that must be done to implement an examine is to have the console force its own value onto the data lines to be loaded into the ACC instead. The value to be used should, of course, be that value the ACC already has, so the ACC won't be clobbered. Any instruction that sends out the ACC onto the data bus will allow the console to latch it for subsequent stuffing back in during the "LDA" or IN"; the "OUT" will serve nicely. Since no extant I/O device should be affected by this operation, I/O device #0 is reserved for use by the console; an "OUT 0" is performed to latch the ACC.

Operations carried out to realize a deposit are, in summary, the forcing of execution of a "STA" (or "OUT") by the console, while the value to be deposited (rather than the value on the CPU's data lines) is sent out on the backplane. An examine is done by first executing an "OUT" to device #0 (to latch the ACC value), followed by an "LDA" (or
"IN") (during which the latched value is fed back into the microprocessor to insure the ACC isn't changed). Display of the contents of the memory location (or I/O device) long enough to be seen by the programmer is accomplished by preventing the instruction execution from being fully completed; the CPU is held at its last machine cycle until a new console operation is to be performed (this is done with deposits also).

It can be seen that if the hardware is given the capability of responding to an "IN 0" by putting the value of the ACC latch out onto the data bus, then the current contents of the ACC may always be ascertained by examining I/O device #0. This is because examining an I/O register causes the ACC latch to be updated with the current ACC value: the latch would then put its value out on the data bus for display if the device register specified were "0".

The ACC value can be changed as desired if the console executes an "IN" (rather than an "OUT") when depositing into I/O register #0.

The forced execution of the various instructions mentioned has the undesired side-effect of causing the microprocessor's PC to be advanced. This is counteracted by latching the PC value (as sent out on the address bus by the CPU) when program execution is suspended by the console; after each examine or deposit a "JMP" is done to the location specified in the PC latch to set the PC straight.

While all of the foregoing sounds rather complicated, the CPU board's data paths are influenced by the existence of the console only slightly, and the amount of extra logic needed there to allow the console to function is minimal. The console has been termed "optional" because
it can be used at any node as desired; the CPU doesn't need it to operate properly.

3. Implementation

Reference to Appendix VII-A shows that the console operations are carried out under control of a shift register FSM having the capability of holding at and jumping to certain states (jumping has priority). Control of the microprocessor in terms of permitting it to cycle through instruction execution is achieved by raising and lowering the "READY" line of the CPU ('CNREADYL', sent to the CPU board, when asserted forces "READY" low). Combinational logic plus four comparators are used to determine whether a breakpoint match exists.

A 16-bit latch (the 'EDC') holds the address used for examines and deposits, and another (the 'PCL') stores the PC value (derived either from the address bus or the panel switches). Another register (the 'ACL') holds either the ACC value or the low eight switches, depending upon whether an examine or deposit is being done.

A bank of multiplexers chooses among the EDC, PCL, ACL and the bit representation of the instruction to be executed ("JMP", "LDA", "STA", "IN", "OUT") according to what portion of the forced instruction the CPU is executing; the data is sent to the CPU board where it is either fed into the microprocessor or sent out on the backplane data buses.

The console panel itself has some amount of logic to drive the display LED's and bounce-eliminate (as necessary) and buffer the switch and push-button signals. Synchronization of the signals is performed back on the console board. Lastly, the address display is derived from the backplane address bus (for program execution) or from the EDC (for 'LOAD
ADDRESS', 'EXAMINE' and 'DEPOSIT' operations).

E. Memory Controller

1. Circular Buffer Philosophy

Two circular buffers are implemented in the upper memory area by the memory controller (MCON): the 'input circular buffer' (ICB) and the 'output circular buffer' (OCB). The ICB holds messages that have come into the node from over the network, while into the OCB are placed messages the network interface is to transmit. The MCON maintains a group of buffer pointers for use in getting information into and out of the buffers. The microprocessor is allowed to read and write any memory locations it chooses, but the network interface (NI) and any peripheral devices capable of doing DMA are permitted to access only those locations at the top and bottom of the buffers.

The 'ISOB', 'IEOB', 'OSOB' and 'OEOB' pointers demark the beginning and end of the ICB and OCB. For each operation possible (reading as well as writing) for both buffers there is a pair of pointers (a "working" pointer and a "holding" pointer): 'IIHP' and IIWP' for writing (by the NI) into the ICB; 'IOHP' and 'IOWP' for reading from the ICB (by the microprocessor or a peripheral device); 'OIHP' and 'OIWP' for writing into the OCB (by CPU or I/O device); and 'OOHP' and 'OOWP' for reading from the OCB (by the NI). All twelve pointers are maintained in the MCON hardware; to permit the software to set the values initially, as well as to read and change (as necessary) the values later on, I/O logic on the MCON board gives the microprocessor full access to the pointers.

As successive words are accessed (by the NI or DMA I/O device) the proper working pointer is incremented, while the holding pointer stays and
marks where the working pointer started from at the beginning of the information transfer. Since the buffers are circular, if the working pointer bumps into the 'EOB' pointer, the 'SOB' value gets loaded back in again. The amount of room in the buffer for writing in data is determined by the circular distance of the -IWP from the -OHP; likewise, when the -OWP hits the -IHP, no more information is available to be taken out of the buffer. In this manner the holding pointer prevents, for example, any part of a message from being looked at if the message is still being received in. The holding pointer is moved to where the working pointer is only after the whole message has been taken care of.

The holding pointers see to it that information effectively enters and leaves the buffers on a message (rather than byte or word) basis. This is important when it is recognized that messages being transmitted over the network may have to be repeated if the destination(s) did not receive the data properly; the holding pointer remembers the start of the message and can be used to properly reset the working pointer if a retransmission is to be done. Similarly, a message being received in may turn out to be bad eventually; the 'IIWP' can be reset to coincide with the 'IIHP' then so that the buffer space can be written over again.

While the intent is to realize two circular buffers to hold all message information, a good deal of latitude exists in exactly what sort of data structure(s) may be used. Several circular buffers may be maintained by properly manipulating the pointers, as long as all DMA devices (including the NI) are incapacitated in so far as their ability
to reference memory is concerned while the pointers are changed by the CPU. By the same token, a "linear" DMA may be done by seeing to it that the end-of-buffer pointer cannot be reached by the working pointer.

2. **Upper Memory Access**

The memory controller and the upper memory must together be capable of a fast enough cycle time so that the network interface is able to send and receive information at the rate of 7 MHz; the upper memory has a width of 16 bits to realize the high bandwidth required. The controller recognizes requests for memory access from the network interface (NI), peripheral DMA devices and the microprocessor, and has a priority scheme of granting service requests to insure that the NI will never have to wait too long to be serviced (the NI has highest priority, followed by DMA I/O devices, with the CPU last). As each access requires 1100 nsec (worst case), the most the NI has to wait is 2200 nsec to read or write a 16-bit word.

The mechanics of placing requests and having them serviced is straightforward. There are three sets of request/grant signals, with each set consisting of 'REQUEST', 'ACKNOWLEDGE' and 'READY' lines. Requests by the NI for service are done similarly to the way peripheral devices access the memory (see Section III.B.3).

The memory controller translates the service requests into the proper sequence of operations to control the upper memory. The 'ACCESS UMEM' upper backplane signal (see Fig. 14) is sent out to indicate an access is desired; 'UMEM BUSY' is returned by the memory board specified by 'UMA14'-'UMA12' while the memory is busy. 'UMEM WRITE' is asserted by the controller if information is to be written; otherwise, it is not
asserted. For accesses to the circular buffers (that is, NI or peripheral DMA requests), 'UMEM WORD ACCESS' is asserted (to indicate a full 16-bit word should be written if a write is to be done). The proper working pointer value is retrieved and sent out into the 'UMAI4'-'UMAI' lines to indicate to the memory the particular location of interest. The NI or DMA device has no part in specifying the address used; likewise, the memory controller is not concerned with the actual data being transferred from or to memory.

A request for access from the microprocessor is handled slightly differently. Logic on the CPU board detects when the software attempts to access a location in the upper half of the address space; the CPU's own set of request-grant signals are then used to ask for a memory access. When the controller grants the request, however, the 'UMEM WORD ACCESS' line is held unasserted; this recognizes the fact that the CPU can write only 8-bit bytes into memory. Also, the controller's tri-state buffers which usually drive 'UMEM WRITE' as well as the 'UMA' lines become disabled; this permits the CPU logic, when the controller's 'ACKNOWLEDGE' signal is seen, to enable its own tri-state gates and thereby indicate to the memory the direction of data flow and also what location is to be accessed.

It was seen (Section III,B.4) that the lower memory cannot put its data onto the data bus (when doing a read) until the microprocessor disables its tri-state buffers (as signaled by the 'LMEM DATA ENABLE' line being asserted "low"). Since there is no reason why the upper memory boards cannot drive the 'UMD' lines whenever 'UMEM WRITE' is not asserted, the signal that would correspond to 'LMEM DATA ENABLE' on the upper back-
plane is held constantly "low". In fact, all non-memory boards use the line as an additional tie point to "ground".

3. **Pointer Manipulation**

   The microprocessor has the ability to read and write all circular buffer pointers. This is useful, of course, when things in the software are being initialized; the location and extent of the buffers can be defined by the beginning- and end-of-buffer pointers, and the holding and working pointers can be properly reset.

   The program would maintain its own set of working pointers for reading and writing data; when the last of the message has been transferred, the proper hardware holding pointer would be updated. This indicates then that a new message has been placed in the OCB or that space has been opened up in the ICB (depending upon which buffer had been accessed).

   When a peripheral device is to do DMA, the working pointer of interest must point to the proper place in memory. The software can start the device operating; the interface will make requests for memory accesses as described previously. [Note that the device interface indicates to the MCON which buffer is to be accessed; the software must be able to specify such information therefore to the device in its control register(s).] If the transfer is not successful (for example, a bad disc data transfer), the working pointer can be reset and another attempt to move the data made. Otherwise, the proper holding pointer would be updated by the software so as to indicate the movement (in or out) of a block of information.

   The network interface's operation, on the other hand, is not under
the direct supervision of the software. Therefore, means must exist in the controller's hardware to move the pointers around automatically. Thus, in addition to the access request, the NI can also request that a pointer be changed [an 'operate' (on pointer) request]. One (non-bussed) line is used to make the request, while a second specifies whether the previous information transfer was okay ('update' the holding pointer) or not ('reset' the working pointer). The 'READY' and 'ACKNOWLEDGE' lines are used just as with making an access request, and the same line that specifies which buffer is to be used for an access (write into the ICB or read from the OCB) is used to resolve which pointer is to be rewritten.

Four flags are maintained by the MCON to record the status of the buffers (ICB/OCB empty/full). The flag values, properly updated when any pointer is changed (see Appendix VIII-D), are made available to the network interface. The NI can thus determine whether a message has exhausted available buffer space (for either writing or reading), and can use the information in determining the disposition of a message. Also, the flags are made available to the software through the NI status registers (as explained in Section III.F), so that message arrival and departure can be monitored.

Care must be taken in transferring pointer information from and to the CPU. Since the microprocessor reads and writes only eight bits at a time, it must be insured that, for example, from the time the low byte of a pointer is read until the software gets around to reading the upper byte, no change has occurred in the pointer's value. The network interface will, as a general rule, always be processing messages and
would indeed cause the pointers to be updated. Hence, requests for pointer information by the CPU (reading or writing) are done on a full word basis only. This means that, when writing, the low byte is latched until the upper byte appears, at which time the whole word gets written in. Similarly, a pointer value read out is held until both bytes have been digested by the program.

It should be obvious that only one pointer access request can be taken care of at a time. Hence, the situation must be avoided where, for instance, a request to read a pointer has been made by the main body of a program, and then an interrupt- (or trap-) servicing routine gets started up which also attempts to read a pointer value. The problem can be gotten around by careful writing of the software, but was felt to be annoying enough to be cured in the hardware. Thus, the memory controller will automatically inhibit interrupts from being serviced from the time the low byte of a pointer is accessed until care is taken of the high byte.

4. Implementation

Requests are received in (Appendix VIII-A) and latched. If the controller is busy, the present cycle is allowed to complete before action is taken on any new requests. Simple combinational logic determines priority, and thus which request is next granted. Preparations are made for activating the proper FSM, and the proper 'ACKNOWLEDGE' signal is asserted.

The gated clock, two cross-coupled monostables, normally lies dormant, but is allowed to operate when a request comes in; it stays on until the request is finished being processed.
Logic interfaced to the backplane address and data lines functions as an I/O device to permit the software to access the pointers. Microprocessor operation is held up while the MCON acts on the request for pointer information. Interrupts are inhibited during the two halves of a pointer transfer to or from the CPU (as explained earlier).

Four FSM's are realized by shift registers. Combinational logic, including two multiplexers, determines whether the memory is to be read or written, and also generates the addresses used to access the local memory in which the pointers are held.

An eight-bit register is used to temporarily hold the lower byte of data when the microprocessor updates a pointer value. Data fed to the pointer memory (consisting of 16 words of 16 bits each) comes either from the temporary register and the backplane data bus or from a 14-bit 'counter-register' (CR) (used to hold pointer values read out from the pointer memory before comparison or incrementation; 14 bits are sufficient to completely specify the memory addresses).

Four comparators are used to determine whether the addressed pointer in the pointer memory matches the value held in the CR. Demultiplexers determine which flag flip-flop is to be affected by pointer operations.

A 14-bit register holds the value of the working pointer of interest for use in addressing the upper memory. Another 14-bit register holds data for transmission to the CPU as a result of an I/O request to read a pointer's value.

F. Network Interface

All aspects of network protocol and message transmission and reception are handled by the network interface (see Chapter II), with the CPU
acting only in a supervisory capacity. The software places messages in the OCB (byte by byte or via a DMA I/O device) and takes care of messages received in from over the network that have been placed in the ICB. When an error condition involving message transmission comes up, the NI will inform the program of the problem; the software must decide what action will then be taken.

The design of circuitry to perform the function of the NI was not the author's responsibility, and is being carried out independently. 9

1. Message Representation in Memory

The format of the messages placed in the OCB for transmission is similar to that shown in Figs. 5, 6 and 8, with the exception that the response field and check bytes, supplied by the NI when the message is sent out over the network, do not appear. Thus, a message consisting of only one data byte would take up five bytes in the OCB (and would be transmitted in nine bytes). All bytes would go out as set up in memory, with the following exceptions:

1) The source byte in the control field has the node's ID number when the message is transmitted; the second byte of the message as it appears in the OCB is completely ignored.

2) The 'S' bit of the 'length-high' byte of the transmitted message (Fig. 7) contains serialization information; the corresponding bit of the 'length-high' byte placed in the OCB is used in conjunction with message retransmission during error recovery (as explained later).

Messages placed in the ICB after reception have been stripped of response field and check bytes; otherwise, all bytes appear as they were transmitted by the source node.
Since the upper memory is accessed by the NI in terms of 16-bit words, all messages exist in memory starting at even-numbered byte addresses. The number of data bytes in the received message is as indicated in the 'length' control bytes; the last byte of memory associated with the message will contain garbage if the transmitted message had an odd number of data bytes.

2. Control and Status Registers

The CPU monitors and controls the operation of the NI through five registers:

1) 'ID' (read-only by the CPU) -- this register (Fig. 16) provides the value of the ID number of the NI to the CPU.

2) 'Control' (write-only by the CPU) -- through this register the microprocessor can order the NI to enter and leave the network (see Fig. 17), and can prevent the NI from accessing the OCB and ICB (message activity is not immediately interrupted; the order to cease activity is not obeyed until the current message, if any, is taken care of).

3) 'Status' (read-only by the CPU) -- bits #6-#3 (Fig. 18) give information regarding the fullness and emptiness of the two circular buffers. Bit #7 is set when additional space has become available in the OCB. Bits #2 and #1 are set in response to the corresponding bits having been set in the 'control' register (bit #1 is also set when the message at the top of the OCB is undeliverable). Bit #0 indicates that a resync condition is (was) found to exist on the network.

4) 'Interrupt enable' (readable and writable by the CPU) -- the bits in this register (see Fig. 19) simply enable the corresponding bits of the 'status' register to cause interrupts (bits #5, #4 and #3 are not used).
5) 'Error' (read-only by the CPU) -- when the message at the top of the OCB cannot be delivered, this register (see Fig. 20) indicates the reason. Bits #6-#1 correspond directly with the same-numbered bits of the message's 'response' byte. Bits #5, #3 and #0 indicate 'terminal errors'; the NI gives up trying to transmit the message immediately (without further attempts at retransmission). Bit #0 indicates that either the memory controller did not process the NI's memory access request quickly enough or that the OCB became empty during the transmission of the message (this causes the source node to institute a resync, as described in Section II.C.5). Bit #7 is asserted if a resync was detected during message transmission.

3. Message Retransmission after an Error

When the source node finds that the top message in the OCB is undeliverable, the reason is relayed to the software via the 'error' register, and bit #1 of the 'status' register is set (note that if the NI has set that bit as the result of bit #5 of the 'control' register having been set, the 'error' register will have "0" in it). The NI will not attempt to transmit any messages as long as bit #1 of the 'status' register is on.

After an error is reported, the software may or may not want to have the NI retransmit the message. If the program wants to "flush" the message and get on with the transmission of others, it need only update the 'OOHP' to point to the next message (if one exists) and then reset bit #1 of the 'status' register (by first setting and then clearing bit #5 of the 'control' register). If, however, the message is to be sent out again, the network interface's 'output' serialization table entry (from
which will come the 'S' bit of the 'length-high' control byte) must be complemented. This is because, when the NI gave up on trying to send the message, the 'output' entry was updated to be ready for a succeeding message. To insure that the serialization bit sent out this time is the same as that used originally, the complement of the entry's present value must thus be used. Bit #4 of the 'length-high' control byte of the message as set up in the memory before transmission is the means whereby this is accomplished; if the bit is asserted, the entry is complemented before being used as the serialization bit.

The exact course of action to be taken by the software depends upon the particular error reported in the 'error' register. If bits #7, #4, #2 or #1 (see Fig. 20) are on, then the most straightforward thing to do is simply to send the message out again. This can be done immediately, or the software can decide to wait before retransmission (perhaps, for example, to allow the destination node's ICB to become less full). To permit transmission of other messages during the waiting interval, it would be possible to have the software transfer the data out of the OCB (and store it someplace else in memory for a while) and then later move the information back to the OCB.

Terminal errors require more involved recovery procedures. For example, if bit #0 of the 'error' register is on, a hardware malfunction may have occurred or the message control field may have been formulated incorrectly (so as to have indicated an excessively long message). Bit #3 of the 'error' register, if on, tends to imply a lack of synchronization between the serialization bits of the source and destination(s). A message sent with the 'U' bit of the 'length-high' control bit on would
be used to set things straight. Finally, a hardware malfunction of some node on the network could only be suspected if bit #5 of the 'error' register is seen.
IV. SOFTWARE OVERVIEW

Once there exists at each node hardware to realize a working processing system (Chapter III) with a specialized I/O interface to handle message transmission and reception over the network (Chapter II), the characteristics of the system as a whole will be determined largely by the software run at the nodes. A very brief discussion is presented here of the sorts of issues that will be addressed by system software in any attempt to achieve interprocessor communication and resource sharing.

A. Message Protocol

1. Message Types

Efficient use of the network implies message lengths as long as practical because of the wait involved in having to gain control of the network for transmission, as well as the fact of the fixed overhead of the control and response bytes in each message. Thus, long 'data' messages of up to thousands of bytes in length will be sent from source to destination nodes.

If the destination can process a message quickly after it is received, then room in the ICB will be freed soon so that other messages can come in. In the more general case of the node taking an appreciable length of time to digest the data, and/or the node constantly being sent messages from several other different nodes, it is inefficient to have source nodes try to send out information whenever they feel like it. Messages will not be lost, of course, if a destination's ICB is too full to accept them, but the source ties up the network by needless retransmissions.
A second type of message, a short 'acknowledgement' message, would be sent by the destination back to the source to signal completion of processing of the last message; the next 'data' message (if any) could then be sent and probably would find enough room in the destination's ICB.

The number and kinds of messages that can be sent out over the network are as many and varied as the software desires; the hardware transmission facility treats them all the same and is unable to distinguish among them. The software might follow the convention of using the first byte of each message to indicate its kind ('data', 'acknowledgement', etc). If it is possible to get by with just four general classes of messages, the 'Fl' and 'F0' bits of the 'length-high' control byte of the message (see Fig. 7) may be set up as desired and interpreted at the destination to indicate what sort of message has arrived in (the hardware ignores the bits and simply delivers them with the rest of the message).

2. **Network Integrity**

   The software may try to ascertain whether the network appears to be functioning correctly, as follows:

   1) Insist that each message received be followed some time by transmission of an 'acknowledgement' message back to the source; lack of receipt of an expected acknowledge within a given length of time will indicate to the source node that a malfunction has occurred.

   2) When the nodes are not busy sending messages of importance, special 'test' messages can be transmitted. Nodes receiving them would be expected to perform a simple arithmetic and/or logical operation on
the data received, and then send the transformed data back to the source. Comparison of the data could be carried out by the source to verify that both the network interfaces and the microprocessors of the nodes involved seem to be functioning correctly.

If a node is suspected of malfunctioning, the microprocessor can be told (typically, via the switch and light panel) to command the network interface to remove itself from the ring. If the network is still unable to operate properly, the node can be completely bypassed by physically disconnecting the transmission lines from the hardware and reconnecting them together to close the circle again.

B. Resource Management

Connection of I/O devices to the network can be realized in two ways (see Fig. 2):

1) The device is interfaced to a computer; the computer has an I/O interface which allows it to communicate data back and forth (over short distances) with the microprocessor of one of the network nodes.

2) The device is interfaced directly to the microprocessor of one of the nodes of the network.

The first type of interconnection (the entire collection of hardware there can be called a 'processor node') is clearly the one usually desired; computer systems already in existence can be brought onto the network merely by giving the system's CPU the ability to exchange information with the microprocessor of the node. In this situation the microprocessor acts as an I/O pre- and post-processor, while the system's CPU must be able to manage all associated I/O devices.
If a good deal of information is being transferred from the device (a disc, for example) out onto the network (or from network to device), the CPU will be spending most of its time attending to the disc and will not be able to run other processes. Or, if an operating system (OS) is not supported, the CPU can only handle the disc software and nothing else. In these cases the I/O device should be connected instead to some node's microprocessor (resulting in a 'peripheral node') so that the otherwise tied-up CPU can be free to do other things.

1. Processor Nodes

Two situations are of interest in the DSL environment:

1) The processor is able to support an OS. The software running on the node should be able to accept blocks of information from the processor for transmission over the network, and should relay incoming data to the OS also in block form. It is seen that the data is merely resting in the node's circular buffers for a relatively short time before being passed on.

2) The processor cannot support an OS. Since relatively simple-minded programs will be run (I/O done via programmed transfers or perhaps interrupts but typically not via DMA), the node microprocessor software will be called upon to hand characters or data words to the processor (and vice versa) one at a time; the buffering of information is taking place only in the microprocessor's memory in this case.

2. Peripheral Nodes

The types of I/O devices may be seen to be two:
1) Devices (such as discs) which can be time-multiplexed among several data sources and sinks ('sharable devices').

2) Devices (such as line printers, modems, paper tape readers, etc.) which can only be used by one process at a time ('nonsharable devices').

Software to manage sharable devices must be able to access stored information using file names, must set up a file directory, must be able to retrieve information through a file system and must address questions of accessing privileges and file security.

Since no process ever has control of a sharable device (in the sense of being able to transfer information in and out) for more than a short time, many problems are not encountered that must be dealt with in the case of nonsharable devices (as discussed next).

3. Nonsharable Device Management

When a processor wants to make use of a network I/O device, the node microprocessor must formulate a short message, sent out to the node managing the resource, asking whether the device is already in use. If not, information exchange may take place, and the requesting node is granted exclusive access to the device. If the device is already in use, the request must be queued up, and is granted only when all other pending requests have been taken care of (unless a priority scheme of granting requests is implemented).

As soon as the processor is finished with data transfer from or to the device, the device manager may grant a pending request for use of the resource. In the case of an OS-supporting processor, the desire to "release" the device will be explicitly communicated to the node micro-
processor software. However, if an OS does not exist, the user must intercede for the processor (when, for instance, the problem program has finished running) and tell the node software (via the switch and light panel) that use is no longer to be made of the network device(s).

It can be seen that the problem of "deadly embrace"\textsuperscript{10} will be encountered when several processors try to grab nonsharable devices that each needs. For example, two computers may both want to use the line printer and paper tape punch (both assumed to be network-accessible). If the first is granted use of the line printer and the second use of the punch, then both processes will be stymied; neither can continue on because they haven't been granted access to all devices needed. Reference 10 discusses some possible solutions to the problem, but in the DSL where most processors happen to be non-OS-supporting minicomputers, the most straightforward approach is to look for user intervention. That is, it will be assumed that if several processors needing the same resources hang up indefinitely waiting for I/O to complete, then the users will direct the node microprocessors to free up whatever devices may have been grabbed so that one program at least may run to completion.

If a large disc exists somewhere on the ring, then non-sharable devices can be made to appear somewhat more sharable by "spooling" information. That is, data can be held temporarily on the disc until the non-sharable device is able to accept it; the device then becomes effectively available for use by more than one process at a time.
V. CONCLUSIONS AND RECOMMENDATIONS

A network of microprocessor-based nodes to realize high speed data transmission has been presented. Mechanisms which permit transfer of data at high frequencies around a closed ring were discussed, and the protocol for network control was gone into. The different hardware assemblies that together constitute a node were examined and, finally, an overview of software-related issues was given.

A prototype network of four nodes is under construction and not yet completed. Hence, it is not possible to report on system performance or the efficiency of resource utilization. However, some topics can be suggested as the basis for future research involving the network once it is working.

A. Research Suggestions

1) Bootstrapped operating systems -- the existence of a disc on the network should mean that processors formerly unable to support an OS due to lack of secondary storage could now be able to provide OS-like services; the types and quality of services renderable depend upon the processor's hardware configuration (size of memory, memory protection, etc.).

2) Network console terminal -- an I/O terminal could be interfaced to the network at one node and system software upgraded to enable users to interrogate the network (individual nodes or all nodes together) regarding operational status. For example, if a processor is hung up for lack of a non-sharable I/O device, information regarding what device is needed as well as the request-queue for that device could be presented at the terminal. Device assignment and deassignment for any particular
node could be requested via keyboard commands. Other commands would define a given I/O device as being up or down (in or out of functional readiness).

3) Expansion of the ring's geographical area -- this would involve running transmission lines to other computers at M.I.T. outside the DSL and setting up microprocessor nodes so as to have additional processors on the ring. Practical difficulties involved with high speed data transmission over moderate or long distances would be explored. Also, use could be made of bidirectional transmission links to get around having to lay pairs of long transmission lines to each isolated distant node.

4) Distributed computation -- situations wherein several processors run different portions of one large program simultaneously and communicate their results could be studied. Since the data transfer rate on the network is high, those problems that involve a good deal of interprocessor communication would benefit most from study here. The maintenance of a large data base of graphical information which is to be displayed at several nodes of the network and must be continually updated is one possible example.

5) Distributed operating system -- migration of processes from processor to processor as the overall workload changes so as to balance each processor's load is the goal (see Refs. 2, 3, 4, 5).

B. Project Status

As of the end of January, 1975, several of each type of logic assembly described herein have been built and are being debugged.
Design has not yet been completed on the network interface. The data preprocessor and data preprocessor exerciser have been demonstrated to work properly; for example, transmission and reception of messages up to approximately 32,800 bits long (at a bit rate of 7 MHz) has been achieved on a one-node test ring for hours at a time with no errors occurring.

Software development for resource sharing and interprocessor communication is under way in the DSL but will not be given high priority until a working prototype ring of four nodes is closer to being realized.
Fig. 1. Network as viewed from a resource.

Fig. 2. Possible resource/node configurations.

Fig. 3. Node structure.
Fig. 4. Synchronous data recovery.
**Fig. 5.** Message format.

**Fig. 6.** Message control field.

**Fig. 7.** Message control field length specification bits.

**Fig. 8.** Message data field.
Fig. 9. Message response field.

Fig. 10. Message response bits specification.

<table>
<thead>
<tr>
<th>Node</th>
<th>Output</th>
<th>Input</th>
<th>Input Init</th>
<th>All</th>
<th>All Init</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Fig. 11. Example serialization table layout.
Fig. 12. Node functional block diagram.

Fig. 13. Backplane layout.
<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
<th>J</th>
<th>K</th>
<th>L</th>
<th>M</th>
<th>N</th>
<th>P</th>
<th>Q</th>
<th>R</th>
<th>S</th>
<th>T</th>
<th>U</th>
<th>V</th>
</tr>
</thead>
<tbody>
<tr>
<td>EDGE A1</td>
<td>EDGE A2</td>
<td>EDGE B1</td>
<td>EDGE B2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>-9</td>
<td>+5</td>
<td>INITL</td>
<td>+5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>IRQD**</td>
<td>IRQL**</td>
<td>UMD11</td>
<td>UMD10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>φ1-SYNC</td>
<td>GND*</td>
<td>ACCESSUMEM</td>
<td>GND*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>IRQL</td>
<td>IRQL</td>
<td>UMD9</td>
<td>UMD8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>A7</td>
<td>A6</td>
<td>UMD7</td>
<td>UMD6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>A5</td>
<td>A4</td>
<td>UMD5</td>
<td>UMD4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>H</td>
<td>A3</td>
<td>A2</td>
<td>UMD3</td>
<td>UMD2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>J</td>
<td>A1</td>
<td>A0</td>
<td>UMD1</td>
<td>UMD0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>K</td>
<td>OUT:WR</td>
<td>INP:DBIN</td>
<td>UMEMBUSY</td>
<td>UMA14</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>L</td>
<td>D7</td>
<td>D6</td>
<td>UMA13</td>
<td>UMA12</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>M</td>
<td>D5</td>
<td>D4</td>
<td>UMA11</td>
<td>UMA10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N</td>
<td>PERIPH ACCESS REQUEST</td>
<td>GND*</td>
<td>UMEM WRITE</td>
<td>GND*</td>
<td>***</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P</td>
<td>D3</td>
<td>D2</td>
<td>UMA9</td>
<td>UMA8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R</td>
<td>D1</td>
<td>D0</td>
<td>UMA7</td>
<td>UMA6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S</td>
<td>PERIPH BUFFER SPEC</td>
<td>PERIPH REQUEST ACKNOWLEDGE</td>
<td>UMA5</td>
<td>UMA4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>GND</td>
<td>PERIPH FSM READY</td>
<td>GND</td>
<td>UMEM WORD ACCESS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>U</td>
<td>UMD15</td>
<td>UMD14</td>
<td>UMA3</td>
<td>UMA2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>V</td>
<td>UMD13</td>
<td>UMD12</td>
<td>UMA1</td>
<td>UMA0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

** Must be explicitly wired onto proto-boards.
*** Serial priority line; not busbed.
**** Memory boards use this pin as a signal (MEM DATA ENABLE) and it is not tied to ground on those boards.

Fig. 14
<table>
<thead>
<tr>
<th>A</th>
<th>-9</th>
<th>+5</th>
<th>+12</th>
<th>+5 *</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td></td>
<td></td>
<td>LMD1</td>
<td>LMD10</td>
</tr>
<tr>
<td>C</td>
<td>GND *</td>
<td>ACCESS</td>
<td>LMEM</td>
<td>GND *</td>
</tr>
<tr>
<td>D</td>
<td></td>
<td></td>
<td>LMD9</td>
<td>LMD8</td>
</tr>
<tr>
<td>E</td>
<td></td>
<td>LMD7</td>
<td>LMD6</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td></td>
<td>LMD5</td>
<td>LMD4</td>
<td></td>
</tr>
<tr>
<td>G</td>
<td></td>
<td>LMD3</td>
<td>LMD2</td>
<td></td>
</tr>
<tr>
<td>H</td>
<td></td>
<td>LMD1</td>
<td>LMD0</td>
<td></td>
</tr>
<tr>
<td>I</td>
<td></td>
<td>LMEM BUSY</td>
<td>LMA14</td>
<td></td>
</tr>
<tr>
<td>J</td>
<td></td>
<td>LMA13</td>
<td>LMA12</td>
<td></td>
</tr>
<tr>
<td>K</td>
<td></td>
<td>LMA11</td>
<td>LMA10</td>
<td></td>
</tr>
<tr>
<td>L</td>
<td>GND *</td>
<td>LMEM WRITE</td>
<td>LMEM DATA ENABLE</td>
<td></td>
</tr>
<tr>
<td>M</td>
<td></td>
<td>LMA9</td>
<td>LMA8</td>
<td></td>
</tr>
<tr>
<td>N</td>
<td></td>
<td>LMA7</td>
<td>LM6</td>
<td></td>
</tr>
<tr>
<td>O</td>
<td></td>
<td>LMA5</td>
<td>LMA4</td>
<td></td>
</tr>
<tr>
<td>P</td>
<td></td>
<td>LMEM WORD ACCESS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q</td>
<td></td>
<td>LMD15</td>
<td>LMD14</td>
<td></td>
</tr>
<tr>
<td>R</td>
<td></td>
<td>LMA3</td>
<td>LMA2</td>
<td></td>
</tr>
<tr>
<td>S</td>
<td></td>
<td>LMD13</td>
<td>LMD12</td>
<td></td>
</tr>
<tr>
<td>T</td>
<td></td>
<td>LMA1</td>
<td>LMA0</td>
<td></td>
</tr>
</tbody>
</table>

**LOWER MEMORY AREA SIGNALS LIST**

* → MUST BE EXPLICITLY WIRED ON PROTO-BOARDS.

Fig. 15
Fig. 16. Network interface ID register.

ID number

Fig. 17. Network interface control register.

Register is reset to zero at power-up.

Fig. 18. Network interface status register.
Fig. 19. Network interface interrupt enable register.

Fig. 20. Network interface error register.
Fig. 21. Data preprocessor.

Fig. 22. Data preprocessor exerciser.
Fig. 23. ROM board.

Fig. 24. RAM board.
Fig. 27. Console.

Fig. 28. Console front panel.
BIBLIOGRAPHY


APPENDIX I

Data Preprocessor Schematic
NOTE: The "10M" and "20M" designations are holdovers from when the basic bit timing was 10 MHz.

* => Special VCC BYPASSING
U1 = 74800
U2 = 74800
U3 = 74800
U4 = 9601
U5 = 74157
U6 = 7402
U7 = 9602
U8 = 74574
U9 = 7451
U10 = 74874
U11 = 74874
U12 = 7474
U13 = 7451
U14 = 74193
U15 = 74193
U16 = 9338
U17 = 9338
U18 = 7474
U19 = 7430
U20 = 74874
U21 = 12061
U22 = 74157
[U23 - UNUSED]
U24 = 4044
U25 = 10125
U26 = 7404
U27 = 741
U28 = 1648
U29 = 7474
<table>
<thead>
<tr>
<th></th>
<th>EDGE C1</th>
<th>EDGE C2</th>
<th>EDGE D1</th>
<th>EDGE D2</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>-9</td>
<td>+5</td>
<td>+12</td>
<td>+5 *</td>
</tr>
<tr>
<td>B</td>
<td></td>
<td></td>
<td>20M CLOCK (OUT)</td>
<td>INIT PG COUNTER</td>
</tr>
<tr>
<td>C</td>
<td>GND</td>
<td>GND</td>
<td>GND</td>
<td>GND</td>
</tr>
<tr>
<td>D</td>
<td></td>
<td></td>
<td>PG LOSS</td>
<td>RESET PG LOSS</td>
</tr>
<tr>
<td>E</td>
<td></td>
<td>CFU RESET</td>
<td>CFU</td>
<td>EXTENDED SIGNAL LOSS</td>
</tr>
<tr>
<td>F</td>
<td>GND</td>
<td>GND</td>
<td>GND</td>
<td>GND</td>
</tr>
<tr>
<td>H</td>
<td></td>
<td></td>
<td>ENCODING ERROR BEFORE PG COMP.</td>
<td></td>
</tr>
<tr>
<td>J</td>
<td></td>
<td>DATA BEFORE PG COMP.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>K</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>L</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>M</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N</td>
<td>GND</td>
<td>GND</td>
<td>GND</td>
<td>GND</td>
</tr>
<tr>
<td>P</td>
<td></td>
<td></td>
<td>DATA TO FSM</td>
<td></td>
</tr>
<tr>
<td>R</td>
<td></td>
<td></td>
<td>ENCODING ERROR TO FSM</td>
<td></td>
</tr>
<tr>
<td>S</td>
<td></td>
<td></td>
<td>10M CLOCK (IN)</td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>GND</td>
<td>GND</td>
<td>GND</td>
<td>GND</td>
</tr>
<tr>
<td>U</td>
<td>LOCK PLL ONTO XTL</td>
<td>RAW DATA IN</td>
<td></td>
<td></td>
</tr>
<tr>
<td>V</td>
<td>10M CLOCK (OUT)</td>
<td>RM</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**DATA PREPROCESSOR SIGNALS LIST**

*⇒ MUST BE EXPLICITLY WIRED ON PROTO-BOARDS.
APPENDIX II-A

Data Preprocessor Exerciser Schematic
NOTE: AS INDICATED, ALL ICs ON THIS PAGE DERIVE VCC FROM A SEPARATE SUPPLY. THE BYPASS CAPACITOR IS INSERTED INTO THE 10-PIN SOCKET ALONG WITH THE IC.
u1 = 7404  
u2 = 7402  
u3 = 7400  
u4 = 7432  
u5 = 7437  
u6 = 7413  
u7 = 7400  
u8 = 7402  
u9 = 7410  
u10 = 9602  
u11 = 9602  
u12 = 7474  
u13 = 7404  
u14 = 7427  
u15 = 7402  
u16 = 74195  
u17 = 74195  
u18 = 74195  
u19 = 7404  
u20 = 7408  
u21 = 7451  
u22 = 7400  
u23 = 7408  
u24 = 74107  
u25 = 74107  
u26 = 74107  
u27 = 9602  
u28 = 7474  
u29 = 7425  
u30 = 7451  
u31 = 7451  
u32 = 7402  
u33 = 74174
<table>
<thead>
<tr>
<th></th>
<th>EDGE C1</th>
<th>EDGE C2</th>
<th>EDGE D1</th>
<th>EDGE D2</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>-9</td>
<td>+5</td>
<td>+12</td>
<td>+5</td>
</tr>
<tr>
<td>B</td>
<td>FAILSAFE</td>
<td>+5</td>
<td>20M CLOCK (OUT)</td>
<td>INIT FG COUNTER</td>
</tr>
<tr>
<td>C</td>
<td>GND*</td>
<td></td>
<td></td>
<td>GND*</td>
</tr>
<tr>
<td>D</td>
<td></td>
<td></td>
<td>FG LOSS</td>
<td>RESET FG LOSS</td>
</tr>
<tr>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td>EXTENDED SIGNAL LOSS</td>
</tr>
<tr>
<td>F</td>
<td></td>
<td>ENCODING ERROR BEFORE FG COMP</td>
<td>DATA BEFORE FG COMP</td>
<td></td>
</tr>
<tr>
<td>G</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>H</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>J</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>K</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>L</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>M</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>N</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>O</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>P</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>S</td>
<td></td>
<td></td>
<td>10M CLOCK (IN)</td>
<td>GND</td>
</tr>
<tr>
<td>T</td>
<td></td>
<td></td>
<td></td>
<td>GND</td>
</tr>
<tr>
<td>U</td>
<td></td>
<td></td>
<td>LOCK PLL ONTO XTAL</td>
<td>RAW DATA IN</td>
</tr>
<tr>
<td>V</td>
<td></td>
<td></td>
<td>10M CLOCK (OUT)</td>
<td>RM</td>
</tr>
</tbody>
</table>

**DATA PREPROCESSOR EXERCISER SIGNALS LIST**

* * * MUST BE EXPLICITLY WIRED ON PROTO-BOARDS.
APENDIX II-B

Data Preprocessor Exerciser State Flowchart
SET LOCK PLL ONTO XTAL FF
BYPASS RING INFO AROUND NODE

SET OUTPUT CLAMP FF

SET LOCK PLL ONTO XTAL FF & DISABLE ESL DETECTION
DISABLE FC LOSS & ENCODING ERROR DETECTION
RESET RM FF
RESET \( T_4 \) ENABLE FF
CLEAR IFSM, OFSM
SET MODE MODE TO ORIGIN-DATE
SET OB Mode TO LOAD
SET ISR MODE TO LOAD
RESET SMF (SENDING MESSAGE FLAG)

(S5OUT)

\[ \text{ESL} \Rightarrow \text{JUMP TO } \gamma_5 \] \( \approx \frac{1}{2} (15) \text{ms} \)

DEFAULT

(S5OUT)

RESET OUTPUT CLAMP FF
SET RM FF
ESL \( \Rightarrow \) JUMP TO \( \gamma_5 \)

DEFAULT

RESET OUTPUT CLAMP FF
23 ms
RESET LOCK PLL ONTO XTAL FF & ENABLE ESL DETECTION
ENCODING ERROR OR DATA=0 BEFORE PG COMP.
OR (RM+PC LOSS) RETRIGGER \( \gamma_5 \) CLEAR PC LOSS & INSEF PG-COUNTERS

A
Synchronization State

(immediately) init. pg-counters & clear xs pg-loss
enable pg-loss+encoding error detection

RMFF ⇒ set node mode to copy; jump to NRM
RMFF ⇒ jump to RM
create new process?

ISO
(data = 0) ⇒ hold state
(data = 1) ⇒ reset T4 enable FF

IS1
(data = 1) ⇒ set isr mode to shift
(data = 0) ⇒ (SMF) ⇒ jump to reSync
(data = 0) ⇒ (SMF) ⇒ (NRL mono on + S6 in) ⇒ retriggert T4 mono ⇒ set T4
enable FF
(data = 0) ⇒ (SMF) ⇒ (NRL mono off + S6 out) ⇒ set SMF
(data = 0) ⇒ (SMF) ⇒ jump to ISO

ISR termination ⇒ hold state
ISR termination ⇒ reset SMF,
set isr mode to load,
retriggert T4 mono &
set T4 enable FF
RM

assert data out
retrigger T4, mono & set T4 enable ff

(o data = 0) => hold state
(o data = 1) => set mode mode to copy

nrm

(smf being set) => hold state
(smf being set) => set mode mode to originate

assert data out
set osr mode to shift

osr termination => hold state
osr termination => set osr mode to load

assert data out
retrigger T4, mono & set T4 enable ff

B
B

5MF ⇒ HOLD STATE
5MF ⇒ SET NODE MODE TO COPY
FIRE NRL MOMO AT END OF STATE

NRM
APPENDIX III

ROM Board Schematic
\[ \begin{align*} 
U_1 &= 9602 \\
U_2 &= 74154 \\
U_3 &= 7442 \\
U_4 &= 74125 \\
U_5 &= 74125 \\
U_6 &= 74125 \\
U_7 &= 1702 \\
U_8 &= 1702 \\
U_9 &= 1702 \\
U_{10} &= 1702 \\
U_{11} &= 1702 \\
U_{12} &= 1702 \\
U_{13} &= 1702 \\
U_{14} &= 1702 \\
U_{15} &= 1702 \\
U_{16} &= 1702 \\
U_{17} &= 1702 \\
U_{18} &= 1702 \\
U_{19} &= 1702 \\
U_{20} &= 1702 \\
U_{21} &= 1702 \\
U_{22} &= 1702 \\
U_{23} &= 74125 \\
U_{24} &= 74125 \\
U_{25} &= 74125 \\
U_{26} &= 74125 \\
U_{27} &= 7402 \\
\end{align*} \]
APPENDIX IV

RAM Board Schematic
U1 = 2102
U2 = 2102
U3 = 2102
U4 = 2102
U5 = 2102
U6 = 2102
U7 = 2102
U8 = 2102
U9 = 2102
U10 = 2102
U11 = 2102
U12 = 2102
U13 = 2102
U14 = 2102
U15 = 2102
U16 = 2102
U17 = 74125
U18 = 74125
U19 = 74125
U20 = 74125
U21 = 74125
U22 = 9602
[U23 - UNUSED]
U24 = 7404
U25 = 7400
U26 = 7420
U27 = 7442
U28 = 7402
U29 = 74125
U30 = 74125
U31 = 74125
U32 = 7432
APPENDIX V-A

Line Printer Interface Schematic
* ⇒ SPECIAL BYPASSING  
CN U8, U12
U1 = 7430
U2 = 7404
U3 = 7402
U4 = 7400
U5 = 7474
U6 = 74125
U7 = 7402
U8 = 7437
U9 = 7474
U10 = 7495
U11 = 7495
U12 = 7437
U13 = 7432
U14 = 7474
U15 = 74125
APPENDIX V-B

Line Printer Interface Registers Description
Control and Status Register: Address BE

Bit #7: 'DONE' - Set by INIT
Reset by loading data
Read-only by CPU

Bit #6: 'INTERRUPT ENABLE' - Reset by INIT
Read/write by CPU

Bit #0: 'ERROR' - Reset by INIT
Set by printer not ready or out of paper
Read-only by CPU

Bits #5, 4, 3, 2, 1 - Not used

Data Register: Address BF

Bits #6 - #0: Character to be printed; sent to printer as soon as loaded into register
Write-only by CPU

Bit #7: Not used

Interrupts: Vector Address 18

Interface interrupts iff Control and Status Register has bit #6 on and either bit #7 or bit #0 on. Service routine must either reset DONE (by sending a new character to the printer) or reset INTERRUPT ENABLE (by writing "0" into bit #6 of the Control and Status Register). Note that the ERROR bit will stay on until the error condition is corrected.
APPENDIX VI-A

CPU Board Schematic
NOTE: 'D7-1D0 are brought down to the lower back-plane and there become 1LM07-1LM00.
(BA14) → :LMA14
(BA13) → :LMA13
(BA12) → :LMA12
(BA11) → :LMA11
(BA10) → :LMA10
(BA9) → :LMA9
(BA8) → :LMA8
(BA7) → :A7
(BA6) → :A6
(BA5) → :A5
(BA4) → :A4
(BA3) → :A3
(BA2) → :A2
(BA1) → :A1
(BA0) → :A0

NOTE: :A7 - :A0
ARE BROUGHT DOWN
TO THE LOWER BACK-
PLANE AND THERE
BECOME :LMA7 - :LMA0.
<p>| ( u_1 ) = 7413 &amp; ( u_{35} ) = 74153 &amp; ( u_{69} ) = 7425 &amp; ( [u_{70}\text{-UNUSED}] ) &amp; ( u_{71} ) = 74125 &amp; ( u_{72} ) = 9316 &amp; ( u_{73} ) = 9316 &amp; ( u_{74} ) = 9316 &amp; ( u_{75} ) = 9316 &amp; ( u_{76} ) = 7485 &amp; ( u_{77} ) = 7485 &amp; ( u_{78} ) = 7485 &amp; ( u_{79} ) = 7485 &amp; ( u_{80} ) = 74155 &amp; ( u_{81} ) = 74125 &amp; ( u_{82} ) = 7474 &amp; ( u_{83} ) = 74125 &amp; ( u_{84} ) = 9316 &amp; ( u_{85} ) = 9316 &amp; ( u_{86} ) = 7474 &amp; ( u_{87} ) = 7404 &amp; ( u_{88} ) = 74155 &amp; ( [u_{89}\text{-UNUSED}] ) |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>-9</td>
<td>15</td>
<td>112</td>
<td>65 *</td>
</tr>
<tr>
<td>B</td>
<td>CPU F3M READY</td>
<td>LMD11</td>
<td>LMD10</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>GND *</td>
<td>LMEM</td>
<td>LMEM</td>
<td>GND *</td>
</tr>
<tr>
<td>D</td>
<td>CDRL</td>
<td>LMD9</td>
<td>LMD8</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>MEPAL Ready</td>
<td>CDRI</td>
<td>CDIZ</td>
<td>GIZ</td>
</tr>
<tr>
<td>F</td>
<td>CDI15 INTERRUPT</td>
<td>CL15</td>
<td>CL14</td>
<td></td>
</tr>
<tr>
<td>G</td>
<td>CDI25</td>
<td>CLOL</td>
<td>CL37</td>
<td>CL22</td>
</tr>
<tr>
<td>H</td>
<td>CMD15L</td>
<td>CMD15L</td>
<td>CL12</td>
<td>CD02</td>
</tr>
<tr>
<td>I</td>
<td>CMD16L</td>
<td>CMD14L</td>
<td>LMEMBUSY</td>
<td>LMA14</td>
</tr>
<tr>
<td>J</td>
<td>CMD17L</td>
<td>CMD13L</td>
<td>LMA13</td>
<td>LMA12</td>
</tr>
<tr>
<td>K</td>
<td>CMD18L</td>
<td>CMD11L</td>
<td>LMA11</td>
<td>LMA10</td>
</tr>
<tr>
<td>L</td>
<td>CMD19L</td>
<td>CMD10L</td>
<td>LMA9</td>
<td>LMA8</td>
</tr>
<tr>
<td>M</td>
<td>CMD20L</td>
<td>GND *</td>
<td>LMEM WRITE</td>
<td></td>
</tr>
<tr>
<td>N</td>
<td>CBA15</td>
<td>LMEM DATA ENABLE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>O</td>
<td>CINT</td>
<td>CWAIT</td>
<td>LMA9</td>
<td>LMA8</td>
</tr>
<tr>
<td>P</td>
<td>CSYNCL</td>
<td>CDBINL</td>
<td>CD7F</td>
<td>CD6F</td>
</tr>
<tr>
<td>Q</td>
<td>CPH1L</td>
<td>CGND1</td>
<td>CD5F</td>
<td>CD4F</td>
</tr>
<tr>
<td>R</td>
<td>GND</td>
<td>CNREADYL</td>
<td>GND</td>
<td>LMEM WORK</td>
</tr>
<tr>
<td>S</td>
<td>LMD15</td>
<td>LMD14</td>
<td>CD3F</td>
<td>CD2F</td>
</tr>
<tr>
<td>T</td>
<td>LMD13</td>
<td>LMD12</td>
<td>CD1F</td>
<td>GIOF</td>
</tr>
</tbody>
</table>

**CPU BACKPLANE SIGNALS LIST**

* * * MUST BE EXPLICITLY WIRED ON PROTO-BOARDS.
APPENDIX VI-B

I/O Port-Number Assignments and Device Register Descriptions
I/O #00 - Reserved for Console
I/O #01 - OUT:

Programmed INIT to I/O devices

IN:

Bit #7:

An interrupt has occurred since the last time the register was read.

Bit #6:

RESET line to CPU has been activated (either manually or at power-up).

Bit #5:

Power-up condition has existed.

Bit #4:

(Not used)

Bits #3 - #0:

Subvector specification for the most recent interrupt -- cleared to "0" by CPU RESET.

Bits #7 - #5 are cleared by reading the register.

I/O #02 - OUT:

Bits #7 - #1:

Lower half of stack limit register.

Bit #0:

Stack overflow interrupt enable.

IN:

Bit #7:

Stack overflow detected -- cleared by reading the register and by INIT.
Bits #6 - #1:

(Undefined)

Bit #0:

Stack overflow interrupt enable.

I/O #03 - OUT:

Bits #7 - #0:

Upper half of stack limit register.

IN:

(Undefined)

Note that CPU RESET clears out the register and turns off the interrupt enable bit. The stack limit address is given by the 15-bit value plus and implicit LSB of "0". A stack overflow occurs when the address used for stack reference by the CPU is less than the stack limit address. The stack overflow interrupt has the highest priority of any "device", and interrupts to location 0.

I/O #04 - OUT:

Display value (latched) on the eight lights of switch and light panel.

IN:

Read the value of the eight switches of the switch and light panel.

I/O #05 - OUT:

Bit #6:

Enable the push-button on the switch and light panel to cause interrupts.
Bits #7, #5 - #0:

Unused

IN:

Bit #7:
Set by push-button being depressed (leading edge);
reset by reading the register and by INIT.

Bit #6:
Push-button interrupt enable.

The push-button uses interrupt vector address 8 and specifies a
subvector of "0000"; it has the highest priority after that of the stack
overflow detection mechanism.

I/O #06 - #0F -  Reserved for CPU-related devices
I/O #10 - #DF -  Miscellaneous I/O devices
I/O #EO - #FF -  Memory controller circular buffer pointers
APPENDIX VII-A

Console Schematic
U1 = 7437    U36 = 74107    U71 = 74157    U106 = 7408
U2 = 7404    U37 = 74107    U72 = 74157    U107 = 7408
U3 = 7432    U38 = 7474     U73 = 74157    U108 = 7404
U4 = 7432    U39 = 7410     U74 = 74157    U109 = 7404
U5 = 7432    U40 = 7425     U75 = 9316     U110 = 7404
U6 = 7432    U41 = 7425     U76 = 9316     U111 = 7404
U7 = 7432    U42 = 7425     U77 = 9316     U112 = 7404
U8 = 7432    U43 = 7454     U78 = 9316     U113 = 7408
U9 = 74195   U44 = 74175    U79 = 74125    U114 = 7408
U10 = 74195  U45 = 7402     U80 = 74125    U115 = 7408
U11 = 74195  U46 = 7485     U81 = 74125    U116 = 7404
U12 = 74195  U47 = 7485     U82 = 74125    
U13 = 74157  U48 = 7485     U83 = 74107    
U14 = 7474    U49 = 7485     U84 = 7400     
U15 = 7402    U50 = 7432     U85 = 7402     
U16 = 7404    U51 = 9316     U86 = 74174    
U17 = 7432    U52 = 9316     U87 = 74174    
U18 = 7400    U53 = 9316     U88 = 7432     
U19 = 7402    U54 = 9316     U89 = 7432     
U20 = 7402    U55 = 74157    U90 = 74157    
U21 = 7400    U56 = 74157    U91 = 74157    
U22 = 7408    U57 = 74157    U92 = 74157    
U23 = 7451    U58 = 74157    U93 = 74157    
U24 = 7425    U59 = 9316     U94 = 74174    
U25 = 7400    U60 = 9316     U95 = 74174    
U26 = 7404    U61 = 9316     U96 = 7432     
U27 = 7400    U62 = 9316     U97 = 7400     
U28 = 7400    U63 = 7425     U98 = 7432     
U29 = 7420    U64 = 7408     U99 = 7432     
U30 = 7432    U65 = 74157    U100 = 74157   
U31 = 7430    U66 = 74157    U101 = 7400     
U32 = 7410    U67 = 9316     U102 = 7404     
U33 = 7427    U68 = 9316     U103 = 7404     
U34 = 7400    U69 = 74125    U104 = 7404     
U35 = 74107   U70 = 74125    U105 = 7408     

<table>
<thead>
<tr>
<th></th>
<th>EDGE C1</th>
<th>EDGE C2</th>
<th>EDGE D1</th>
<th>EDGE D2</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>-9</td>
<td>+5</td>
<td>+12</td>
<td>+5*</td>
</tr>
<tr>
<td>B</td>
<td>MCBUSY</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>CMCGND</td>
<td>GND*</td>
<td></td>
<td>GND*</td>
</tr>
<tr>
<td>D</td>
<td>CINHIL</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>CRESETL</td>
<td></td>
<td>CD7Z</td>
<td>CD6Z</td>
</tr>
<tr>
<td>F</td>
<td>INHIBIT</td>
<td>CINITL</td>
<td>CD5Z</td>
<td>CD4Z</td>
</tr>
<tr>
<td></td>
<td>INTERRUPTS</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>H</td>
<td>CPU.RESET</td>
<td>CDNL</td>
<td>CD3Z</td>
<td>CD2Z</td>
</tr>
<tr>
<td>J</td>
<td>CMD17L</td>
<td>CMD16L</td>
<td>CD1Z</td>
<td>CD0Z</td>
</tr>
<tr>
<td>K</td>
<td>CMD15L</td>
<td>CMD14L</td>
<td></td>
<td>LMA14</td>
</tr>
<tr>
<td>L</td>
<td>CMD13L</td>
<td>CMD12L</td>
<td>LMA13</td>
<td>LMA12</td>
</tr>
<tr>
<td>M</td>
<td>CMD11L</td>
<td>CMD10L</td>
<td>LMA11</td>
<td>LMA10</td>
</tr>
<tr>
<td>N</td>
<td>CBA15</td>
<td>GND*</td>
<td></td>
<td></td>
</tr>
<tr>
<td>P</td>
<td>CINTEL</td>
<td>CWAIT</td>
<td>LMA9</td>
<td>LMA8</td>
</tr>
<tr>
<td>R</td>
<td>CSYNCL</td>
<td>CD8INL</td>
<td>CD7F</td>
<td>CD6F</td>
</tr>
<tr>
<td>S</td>
<td>CPH11L</td>
<td>CGND1</td>
<td>CD5F</td>
<td>CD4F</td>
</tr>
<tr>
<td>T</td>
<td>GND</td>
<td>CNREADYL</td>
<td>GND</td>
<td></td>
</tr>
<tr>
<td>U</td>
<td></td>
<td></td>
<td>CD3F</td>
<td>CD2F</td>
</tr>
<tr>
<td>V</td>
<td></td>
<td></td>
<td>CD1F</td>
<td>CD0F</td>
</tr>
</tbody>
</table>

**CONSOLE BACKPLANE SIGNALS LIST**

* * * MUST BE EXPLICITLY WIRED ON PROTO-BOARDS.*
APPENDIX VII-B

Console FSM Flowchart
* ON POWER UP:
  CLEAR EDC
  RESET LEORD FF
  RESET ADRFF
  RESET CONSOLE ACCESS FF

1

HOLD IF SYNC* INTA* MA
JUMP (13 IF SYNC* INTA* MA* BREAK)

13

ASSERT NOTRDY
SET DISABLE INTERRUPTS FF
SET CONSOLE ACCESS FF
ADRFF → CPU DISPLAY MODE FF

2

RESET ADRFF
HOLD IF BUTTON3* LEORD

3

RESET LEORDFF IF BUTTON3* EORD
IF LOAD ADDRESS:
  LOAD SW IN EDC CNTR
  SET ADRFF
  RESET BREAKFF
  JUMP (4)

4

IF LOAD PC:
  LOAD SW IN PC LATCH
  JUMP (10)

10

IF CONTINUE:
  JUMP (11)

11

IF EORD:
  COUNT EDC CNTR IF
  LEORD* MEM* (KEY=EFF)

  HOLD UNTIL BUTTON3

5

EKEY→ EFF
SET LEORD FF
RESET CPU DISPLAY MODE FF
RESET NOTRDY IF EKEY

HOLD IF SYNC* EFF
(ALLOWS FETCH CYCLE OF FAKEED "OUT 0" INST TO COMPLETE ["OUT 0" DONE ONLY FOR EXAM])
PLACE ZERO ON CPU'S DATA LINES
FORCE MUXCNTR TO "UPPER" STATE
(NORMALLY WOULD SKIP TO "ACC"
STATE WHICH WOULD CAUSE NORTDY
TO BE SET TOO SOON IN STATE 7)

HOLD IF SYNC·EFF
(SEND 0 TO CPU FOR 2ND CYCLE
OF "OUT 0" FAKE D INST
[FOR EXAM ONLY])

IF EFF THEN STORE CPU'S DATA IN ACC
LATCH (BY DEFEATING CONSOLE
BUS-OVERRIDE)
ELSE STORE SW IN ACC LATCH
RESET NORTDY

BIT 3 OF INST = EFF+ [EN]- [LOWBYTE EDC·6]
BIT 5 OF INST = MEM

WHEN MUXCNTR = "ACC" SET NORTDY
(AT THE END OF THE INST)
HOLD UNTIL MUXCNTR = "ACC" AND BUTTONS

HOLD UNTIL BUTTONS
(TO DISPLAY INFO ON LIGHTS)
RESET NORTDY
HOLD UNTIL SYNC
10

RESET NOTRDY
RESET CPU DISPLAY MODE FF
BIT 4 OF INST= 0

HOLD UNTIL JUMP
JUMP 0 WHEN [MUXCNT="UPPER"], SYNCHRONIZATION
(FAKES A "JMP" INSTRUCTION TO LOAD CPU'S PC)

11

RESET NOTRDY
SET CPU DISPLAY MODE FF
RESET CONSOLE ACCESS FF
(IN PREPARATION FOR PROGRAM INSTRUCTION EXECUTION)

JUMP 16 IF CONSOLEACCESS SW4 IS SET
(DON'T JUMP IF SINGLE CYCLE)

12

HOLD UNTIL SYNCHRONIZATION

JUMP 0 IF SYNCHRONIZATION, M1, INTA
( TO ALLOW OTHER BUTTONS TO WORK IN BETWEEN INSTRUCTIONS, BUT NOT IN BETWEEN CYCLES OF A SINGLE INSTRUCTION)

13

SET NOTRDY
SET DISABLE INTERRUPTS FF
SET CONSOLE ACCESS FF

HOLD UNTIL CONTINUE BUTTON IS RELEASED

14

HOLD UNTIL JUMP
JUMP 14 WHEN CONTINUE IS PRESSED
NOTE ON THE MUXCNTR:

THE MUXCNTR IS USED TO CONTROL THE SELECT LINES TO MULTIPLEXERS THAT SEND OUT "FAKED" INSTRUCTION INFORMATION TO THE CPU.

<table>
<thead>
<tr>
<th>STATE</th>
<th>MCB</th>
<th>MCA</th>
<th>NAME</th>
<th>DATA OUT</th>
<th>STATES ARE IN THEIR COUNTING ORDER. AN M1 CYCLE ALWAYS CLEARS THE CNTR.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0</td>
<td></td>
<td></td>
<td>INS</td>
<td>INST</td>
<td></td>
</tr>
<tr>
<td>0 1</td>
<td></td>
<td></td>
<td>LOWER</td>
<td>PCL OR EDC LOW BYTE</td>
<td></td>
</tr>
<tr>
<td>1 0</td>
<td></td>
<td></td>
<td>UPPER</td>
<td>PCL OR EDC HIGH BYTE</td>
<td></td>
</tr>
<tr>
<td>1 1</td>
<td></td>
<td></td>
<td>ACC</td>
<td>ACC LATCH</td>
<td></td>
</tr>
</tbody>
</table>

FOR "IN" OR "OUT" INSTRUCTIONS "UPPER" WILL BE SKIPPED (EXCEPT IN STATE 5 AS NOTED).

FOR "JMP" INSTRUCTIONS THE PCL IS SENT OUT, OTHERWISE THE EDC.
APPENDIX VII-C

Console Panel Layout
APPENDIX VIII-A

Memory Controller Schematic
<table>
<thead>
<tr>
<th>U1 = 74107</th>
<th>U34 = 7425</th>
<th>U67 = 9316</th>
</tr>
</thead>
<tbody>
<tr>
<td>U2 = 7474</td>
<td>U35 = 7402</td>
<td>U68 = 9316</td>
</tr>
<tr>
<td>U3 = 7474</td>
<td>U36 = 7400</td>
<td>U69 = 9316</td>
</tr>
<tr>
<td>U4 = 7474</td>
<td>U37 = 7486</td>
<td>U70 = 74125</td>
</tr>
<tr>
<td>U5 = 7474</td>
<td>U38 = 9316</td>
<td>U71 = 74125</td>
</tr>
<tr>
<td>U6 = 74107</td>
<td>U39 = 74125</td>
<td>U72 = 74125</td>
</tr>
<tr>
<td>U7 = 9601</td>
<td>U40 = 7404</td>
<td>U73 = 9316</td>
</tr>
<tr>
<td>U8 = 9601</td>
<td>U41 = 7400</td>
<td>U74 = 9316</td>
</tr>
<tr>
<td>U9 = 74107</td>
<td>U42 = 7432</td>
<td>U75 = 9316</td>
</tr>
<tr>
<td>U10 = 7404</td>
<td>U43 = 9316</td>
<td>U76 = 9316</td>
</tr>
<tr>
<td>U11 = 7402</td>
<td>U44 = 9316</td>
<td>U77 = 74125</td>
</tr>
<tr>
<td>U12 = 7402</td>
<td>U49 = 7451</td>
<td>U78 = 74125</td>
</tr>
<tr>
<td>U13 = 7404</td>
<td>U46 = 7451</td>
<td>U79 = 74125</td>
</tr>
<tr>
<td>U14 = 7437</td>
<td>U47 = 7451</td>
<td>U80 = 74125</td>
</tr>
<tr>
<td>U15 = 7400</td>
<td>U48 = 7451</td>
<td>U81 = 74155</td>
</tr>
<tr>
<td>U16 = 74125</td>
<td>U49 = 7451</td>
<td>U82 = 74155</td>
</tr>
<tr>
<td>U17 = 7402</td>
<td>U50 = 7451</td>
<td>U83 = 74155</td>
</tr>
<tr>
<td>U18 = 7425</td>
<td>U51 = 7451</td>
<td>U84 = 7410</td>
</tr>
<tr>
<td>U19 = 7410</td>
<td>U52 = 3101A</td>
<td>U85 = 7410</td>
</tr>
<tr>
<td>U20 = 7420</td>
<td>U53 = 3101A</td>
<td>U86 = 7404</td>
</tr>
<tr>
<td>U21 = 7451</td>
<td>U54 = 3101A</td>
<td>U87 = 7402</td>
</tr>
<tr>
<td>U22 = 7432</td>
<td>U55 = 3101A</td>
<td>U88 = 7400</td>
</tr>
<tr>
<td>U23 = 7404</td>
<td>U56 = 9316</td>
<td></td>
</tr>
<tr>
<td>U24 = 7402</td>
<td>U57 = 9316</td>
<td></td>
</tr>
<tr>
<td>U25 = 7400</td>
<td>U58 = 9316</td>
<td></td>
</tr>
<tr>
<td>U26 = 7410</td>
<td>U59 = 9316</td>
<td></td>
</tr>
<tr>
<td>U27 = 9316</td>
<td>U60 = 7485</td>
<td></td>
</tr>
<tr>
<td>U28 = 9316</td>
<td>U61 = 7485</td>
<td></td>
</tr>
<tr>
<td>U29 = 9316</td>
<td>U62 = 7485</td>
<td></td>
</tr>
<tr>
<td>U30 = 9316</td>
<td>U63 = 7474</td>
<td></td>
</tr>
<tr>
<td>U31 = 9316</td>
<td>U64 = 7485</td>
<td></td>
</tr>
<tr>
<td>U32 = 74157</td>
<td>U65 = 7404</td>
<td></td>
</tr>
<tr>
<td>U33 = 74157</td>
<td>U66 = 9316</td>
<td></td>
</tr>
<tr>
<td>EDGE[C1]</td>
<td>EDGE[C2]</td>
<td>EDGE[D1]</td>
</tr>
<tr>
<td>-----------</td>
<td>-----------</td>
<td>-----------</td>
</tr>
<tr>
<td>A</td>
<td>-9</td>
<td>+5</td>
</tr>
<tr>
<td>B</td>
<td>NODE BUFFER SPEC</td>
<td>NODE ACCESS REQUEST</td>
</tr>
<tr>
<td>C</td>
<td>GND*</td>
<td>NODE UPDATE SPEC</td>
</tr>
<tr>
<td>D</td>
<td>NODE UPDATE SPEC</td>
<td>MEMCON NOT READY</td>
</tr>
<tr>
<td>E</td>
<td>NODE REQUEST ACKNOWLEDGED</td>
<td>CPU RESET</td>
</tr>
<tr>
<td>F</td>
<td>NODE FSM READY</td>
<td>CMGOND</td>
</tr>
<tr>
<td>H</td>
<td>ICB FULL</td>
<td>OCB FULL</td>
</tr>
<tr>
<td>J</td>
<td>ICB EMPTY</td>
<td></td>
</tr>
<tr>
<td>K</td>
<td>OCB FULL</td>
<td></td>
</tr>
<tr>
<td>L</td>
<td>OCB EMPTY</td>
<td></td>
</tr>
<tr>
<td>M</td>
<td></td>
<td></td>
</tr>
<tr>
<td>N</td>
<td>GND*</td>
<td></td>
</tr>
<tr>
<td>P</td>
<td></td>
<td></td>
</tr>
<tr>
<td>R</td>
<td></td>
<td></td>
</tr>
<tr>
<td>S</td>
<td></td>
<td></td>
</tr>
<tr>
<td>T</td>
<td>GND</td>
<td>GND</td>
</tr>
<tr>
<td>U</td>
<td></td>
<td></td>
</tr>
<tr>
<td>V</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**MEMORY CONTROLLER BACKPLANE SIGNALS LIST**

*⇒ MUST BE EXPLICITLY WIRED ON PROTO-BOARDS.
APPENDIX VIII-B

Memory Controller FSM Flowchart
GET WORKING POINTER FROM THE REGISTER MEMORY AND HOLD IN COUNTER REGISTER
(IIWP/OOWP; IOWP/OIWP)

COMPARE COUNTER REGISTER AGAINST EOB POINTER
(IEOB/OEOB; IEOB/OEOB)

IF UNEQUAL, INCREMENT COUNTER REGISTER
IF EQUAL, LOAD COUNTER REGISTER WITH SOB VALUE
(ISOB/OSOB; ISOB/OSOB)

WRITE COUNTER REGISTER CONTENTS BACK INTO WORKING POINTER LOCATION
(IIWP/OOWP; IOWP/OIWP)

COMPARE COUNTER REGISTER AGAINST APPROPRIATE HOLDING POINTER
(IOHP/OIH; ILHP/OOHP)

CONDITIONALLY SET PROPER FLAG ACCORDING TO THE PARTICULAR ACCESS (SEE FLAG SUMMARY SHEETS)
GET POINTER FROM REGISTER MEMORY AND HOLD IN COUNTER REGISTER
(IHP/OOMP; IUP/00WP)
RESET REQUEST: IC0B/0CB; UPDATE REQUEST: IC0B/0CB

WRITE COUNTER REGISTER CONTENTS INTO PROPER POINTER
(IUP/00WP; IHP/00HB)
AND RESET PROPER FLAG (SEE FLAG SUMMARY SHEETS)

*FN FSM IS USED TO UPDATE AND RESET POINTERS FOR THE NETWORK INTERFACE*
FOR CPU READ (via I/O INST) OF THE LSBYTE OF A POINTER, ACCESS THE REGISTER MEMORY AND HOLD THE VALUE IN A TEMPORARY REGISTER.

FOR I/O OUTPUT OF MSBYTE OF A POINTER, COMBINE WITH (PREVIOUSLY) LATCHED LSBYTE AND WRITE VALUE INTO REGISTER MEMORY.

POINTER REFERENCED IS SPECIFIED BY 4 BITS OF I/O PORT #.

READ POINTER (SPECIFIED BY CPU) BACK INTO THE COUNTER REGISTER.

RET PROPER FLAG (SEE FLAG SUMMARY SHEETS), AND COMPARER COUNTER REGISTER VALUE AGAINST POINTER OF INTEREST.

IF A WRITE WAS DONE (FU1) AND COUNTER REGISTER MATCHES POINTER, SET PROPER FLAG (SEE FLAG SUMMARY SHEETS).

FU FSM IS USED TO ALLOW THE CPU TO ACCESS POINTER VALUES VIA "IN" & "OUT" INSTRUCTIONS.

(FU FSM PERFORMS UPPPER MEMORY ACCESSES FOR THE CPU; BECAUSE OF ITS TRIVIAL NATURE IT IS NOT DRAWN HERE.)
APPENDIX VIII-C

Circular Buffer Pointer Table
I/O Port-Number: 111XXXXB

<table>
<thead>
<tr>
<th>X'XXX</th>
<th>Pointer</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>OEOB</td>
</tr>
<tr>
<td>0001</td>
<td>OSOB</td>
</tr>
<tr>
<td>0010</td>
<td>-</td>
</tr>
</tbody>
</table>
| 0011  | -       | B: 0 → Least significant byte
| 0100  | OOWP    |
| 0101  | OOHP    | 1 → Most significant byte
| 0110  | OIWP    |
| 0111  | OIHP    |
| 1000  | IEOB    |
| 1001  | ISOB    |
| 1010  | -       |
| 1011  | -       |
| 1100  | IOWP    |
| 1101  | IOHP    |
| 1110  | IIWP    |
| 1111  | IIIHP   |

Bit #15 and bit #0 of the values written into the pointers are ignored and assumed to be "1" and "0" respectively. This composite 16-bit value is what gets returned when a read is done. Both bytes of a pointer are read (or written) simultaneously and are therefore proper instantaneous values, even if the network interface happens to be in operation concurrently.

Interrupts become automatically inhibited from one LSB access until the next MSB access. That is, interrupts become inhibited* on an LSB access (read or write) and are reenabled* by an MSB access (read or write).
Reading an LSB → fetch whole word, return LSB, and
   internally latch MSB

Reading an MSB → fetch latched MSB

Writing an LSB → internally latch LSB

Writing an MSB → combine latched LSB with new data and
   write whole word

* XXXX bits of a number are ignored for this operation.

Note that separate latches are used for read and write.
APPENDIX VIII-D

Circular Buffer Flag Operations Summary
Node Operate Request

RESET: resets ICB FULL or OCB EMPTY
UPDATE: resets ICB EMPTY or OCB FULL

Access Request

<table>
<thead>
<tr>
<th>NODE</th>
<th>PERIPHERAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>IIWP = IOHP + set ICB FULL</td>
<td>IOWP = IIHP + set ICB EMPTY</td>
</tr>
<tr>
<td>OOWP = OIHP + set OCB EMPTY</td>
<td>OIWP = OOHOP + set OCB FULL</td>
</tr>
</tbody>
</table>

Note that the working pointer value used is that which the pointer has after having accessed memory.

CB Register Request

A) If writing into IIWP then reset ICB FULL
- IIHP  ICB EMPTY
- IOWP  ICB EMPTY
- IOHP  ICB FULL
- OIWP  OCB FULL
- OIHP  OCB EMPTY
- OOWP  OCB EMPTY
- OOHOP OCB FULL

B) Compare new IIHP with IOHP; if they match, then set ICB FULL
- IOHP  IIHP  ICB EMPTY
- OIHP  OOHHP OCB FULL
- OOHP  OIHP  OCB EMPTY
<table>
<thead>
<tr>
<th>CLEARED</th>
<th>SET</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ICB FULL</strong></td>
<td></td>
</tr>
<tr>
<td>1) node RESET request</td>
<td>1) after node access if IIWP = IOHP</td>
</tr>
<tr>
<td>2) CB Pointer write</td>
<td>2) after CB Pointer write if</td>
</tr>
<tr>
<td>for IIWP or IOHP</td>
<td>new IIHP = IOHP</td>
</tr>
<tr>
<td><strong>ICB EMPTY</strong></td>
<td></td>
</tr>
<tr>
<td>1) node UPDATE request</td>
<td>1) after device access if IOWP = IIHP</td>
</tr>
<tr>
<td>2) CB Pointer write</td>
<td>2) after CB Pointer write if</td>
</tr>
<tr>
<td>for IIHP or IOWP</td>
<td>new IOHP = IIHP</td>
</tr>
<tr>
<td><strong>OCB FULL</strong></td>
<td></td>
</tr>
<tr>
<td>1) node UPDATE request</td>
<td>1) after device access if OIWP = OOHPI</td>
</tr>
<tr>
<td>2) CB Pointer write</td>
<td>2) after CB Pointer write if</td>
</tr>
<tr>
<td>for OIWP or OOHPI</td>
<td>new OIHP = OOHPI</td>
</tr>
<tr>
<td><strong>OCB EMPTY</strong></td>
<td></td>
</tr>
<tr>
<td>1) node RESET request</td>
<td>1) after node access if OOWP = OIHP</td>
</tr>
<tr>
<td>2) CB Pointer write</td>
<td>2) after CB Pointer write if</td>
</tr>
<tr>
<td>for OIHP or OOWP</td>
<td>new OOHPI = OIHP</td>
</tr>
</tbody>
</table>
APPENDIX IX

Integrated Circuit Pin-Number Specifications
7495

7442

74154

7408

23,22,12 & ALL TIED TOGETHER

NOTE: NO CONNECTION TO GROUND.
IC'S USED ONLY IN DATA PREPROCESSOR

4.044

10125

741

1648