A VERSATILE LOCAL AREA NETWORK

BY

DAVID EDMUND WICKLIFF

B.S., University of Illinois, 1984

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Electrical Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 1986

Urbana, Illinois
Abstract

A local area network (LAN) can be defined as an information transport system for data transfer among terminals, computer systems and peripheral equipment, via an interconnecting medium within the confines of a single building or cluster of buildings. The research developed a LAN system suitable for use in a teaching laboratory environment. The LAN system is versatile enough to allow easy connection of many different types of equipment while being relatively inexpensive. The network consists of microprocessor-based protocol interface units connected to a common twisted-pair cable. Each interface unit can support an asynchronous RS-232-C serial interface from user equipment into the LAN with standard speeds up to 9.6k bits per second. The interface units provide virtual circuit service for the attached user equipment via a CSMA/CD technique. The data rate on the network cable is 62.5k bits per second, thus allowing several simultaneous, independent virtual circuits on the network.
Acknowledgements

Partial support for the research was provided by the Corning Foundation Instructional Assistance Award received in the spring of 1984. I would like to express my appreciation to Intel Corporation for the donation of development tools, without which this research project would have been impossible. I would also like to thank independent study students Edith Bergstrand, Jack Hoeflich, Gordon Obrecht and Duane Dinschel for their invaluable assistance during the course of the research. I would like to give special thanks to Professor Ricardo Uribe and his creative environment, the Advanced Digital Systems Laboratory, which made this and many other projects possible.
## Table of Contents

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1.1</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td>2.1</td>
<td>8</td>
</tr>
<tr>
<td>2.2</td>
<td>10</td>
</tr>
<tr>
<td>2.3</td>
<td>11</td>
</tr>
<tr>
<td>2.3.1</td>
<td>13</td>
</tr>
<tr>
<td>2.3.2</td>
<td>13</td>
</tr>
<tr>
<td>2.3.3</td>
<td>15</td>
</tr>
<tr>
<td>2.3.4</td>
<td>16</td>
</tr>
<tr>
<td>2.4</td>
<td>17</td>
</tr>
<tr>
<td>2.4.1</td>
<td>18</td>
</tr>
<tr>
<td>2.4.2</td>
<td>19</td>
</tr>
<tr>
<td>2.4.3</td>
<td>25</td>
</tr>
<tr>
<td>2.5</td>
<td>27</td>
</tr>
<tr>
<td>2.5.1</td>
<td>28</td>
</tr>
<tr>
<td>2.5.2</td>
<td>29</td>
</tr>
<tr>
<td>2.5.3</td>
<td>31</td>
</tr>
<tr>
<td>3</td>
<td>32</td>
</tr>
<tr>
<td>3.1</td>
<td>32</td>
</tr>
<tr>
<td>3.1.1</td>
<td>32</td>
</tr>
<tr>
<td>3.1.2</td>
<td>35</td>
</tr>
<tr>
<td>3.1.3</td>
<td>36</td>
</tr>
<tr>
<td>3.1.4</td>
<td>36</td>
</tr>
<tr>
<td>3.1.5</td>
<td>37</td>
</tr>
<tr>
<td>3.1.6</td>
<td>38</td>
</tr>
<tr>
<td>3.2</td>
<td>40</td>
</tr>
<tr>
<td>3.2.1</td>
<td>40</td>
</tr>
</tbody>
</table>
4 The ADSL LAN Architecture

4.1 Physical Layer
   4.1.1 Character Format
   4.1.2 BitTiming
   4.1.3 Electrical Aspects

4.2 Data Link Layer
   4.2.1 PacketTiming
   4.2.2 ErrorDetection
   4.2.3 Collision Avoidance, Detection and Recovery
   4.2.4 Implementation

4.3 Network Layer
   4.3.1 Packet and Acknowledgement Formats
   4.3.2 Routing
   4.3.3 Packet/Acknowledgement Transaction
   4.3.4 Error Recovery
   4.3.5 Implementation

4.4 User, Command, and Transport Layers

5 Conclusions

Appendices

A Hardware Schematics and Components
B Hardware Diagnostic Software
C User Layer Module
D Transport Layer Module
E Command Layer Module
F Command Layer Command Routines Module
G Network Layer Module
H Data Link Layer Module
I Timer Server Module
References Cited
### List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1 User Status LED Display Sequence after Reset</td>
<td>10</td>
</tr>
<tr>
<td>2.2 DIP Switch Attribute Settings</td>
<td>12</td>
</tr>
<tr>
<td>2.3 RS-232-C Electrical Specifications</td>
<td>13</td>
</tr>
<tr>
<td>2.4 User Interface Pin Assignments</td>
<td>14</td>
</tr>
<tr>
<td>4.1 RS-485 Electrical Specifications</td>
<td>46</td>
</tr>
<tr>
<td>4.2 Data Link State Variables</td>
<td>54</td>
</tr>
<tr>
<td>A.1 Component Table</td>
<td>69</td>
</tr>
</tbody>
</table>
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1</td>
<td>2</td>
</tr>
<tr>
<td>1.2</td>
<td>3</td>
</tr>
<tr>
<td>1.3</td>
<td>5</td>
</tr>
<tr>
<td>2.1</td>
<td>9</td>
</tr>
<tr>
<td>2.2</td>
<td>15</td>
</tr>
<tr>
<td>2.3</td>
<td>18</td>
</tr>
<tr>
<td>2.4</td>
<td>19</td>
</tr>
<tr>
<td>2.5</td>
<td>21</td>
</tr>
<tr>
<td>2.6</td>
<td>24</td>
</tr>
<tr>
<td>2.7</td>
<td>28</td>
</tr>
<tr>
<td>3.1</td>
<td>33</td>
</tr>
<tr>
<td>3.2</td>
<td>39</td>
</tr>
<tr>
<td>4.1</td>
<td>43</td>
</tr>
<tr>
<td>4.2</td>
<td>45</td>
</tr>
<tr>
<td>4.3</td>
<td>48</td>
</tr>
<tr>
<td>4.4</td>
<td>49</td>
</tr>
<tr>
<td>4.5</td>
<td>51</td>
</tr>
<tr>
<td>4.6</td>
<td>55</td>
</tr>
<tr>
<td>4.7</td>
<td>56</td>
</tr>
<tr>
<td>4.8</td>
<td>58</td>
</tr>
<tr>
<td>4.9</td>
<td>59</td>
</tr>
<tr>
<td>4.10</td>
<td>59</td>
</tr>
<tr>
<td>A.1</td>
<td>70</td>
</tr>
<tr>
<td>A.2</td>
<td>71</td>
</tr>
<tr>
<td>A.3</td>
<td>72</td>
</tr>
<tr>
<td>A.4</td>
<td>73</td>
</tr>
<tr>
<td>A.5</td>
<td>74</td>
</tr>
<tr>
<td>A.6</td>
<td>75</td>
</tr>
</tbody>
</table>

viii
Chapter 1

Introduction

Over the past few years, there has been a proliferation of personal (or single user) computer systems and related peripheral equipment (e.g., printers and modems). As a result, there is an increasing need for the ability to share information among these relatively autonomous computer systems and to share peripherals among these computer systems. A straightforward method which allows information and equipment sharing is to connect a cable between every pair of devices you would possibly ever wish to interconnect (Figure 1.1). However, this brute force point-to-point method can quickly become impractical. As the number of devices increases, the cost and routing complexity of laying all the interconnecting cables increase. In addition, problems of incompatible communication interfaces on these devices arise. A more comprehensive and flexible interconnecting method is desirable.

A local area network (LAN) can be defined as an information transport system for data transfer among terminals, computer systems and peripheral equipment, via an interconnecting medium within the confines of a single building or cluster of buildings (Figure 1.2). A logical point-to-point connection (or virtual circuit) could be
Figure 1.1: A Point-to-point Network
Figure 1.2: A Local Area Network
established through a LAN system which is physically connected to all the devices. The circuit is a logical structure supported by the LAN, rather than a direct physical connection, and hence is much more flexible. Notice, however, that this approach would make the network's operation dependent on the fault tolerance of the central LAN system.

The author's research developed a virtual circuit-oriented LAN system called the Advanced Digital Systems Laboratory LAN (ADSL LAN). The ADSL LAN is designed to meet several goals:

- **Low Cost**: The initial cost of the LAN system should be low, as also should be the incremental costs of connecting devices to the LAN system.

- **Versatile User Interface**: The communication interface presented to the user of the LAN should be flexible, allowing a wide variety of devices to be connected.

- **Fault Tolerant**: The LAN should provide reliable communication circuits even under partial failure of the LAN system.

- **Maintainable**: System failures of the LAN should be easily repairable.

### 1.1 Overview of the ADSL LAN Structure

The physical interconnection of the ADSL LAN is based on a bus structure. All user devices connect to the LAN at protocol interface units, or nodes, which are in turn connected to a common cable bus (Figure 1.3). Note that the bus is a broadcast medium on which all nodes receive all transmissions. The node units are responsible
Figure 1.3: The ADSL Local Area Network
for the establishment, maintenance, and disconnection of virtual circuits through the shared use of the bus.

There is no one central component of the ADSL LAN responsible for the control of the shared access to the bus. Instead, the responsibility is distributed among all the node units by the use of a control technique called Carrier Sense Multiple Access with Collision Detection (CSMA/CD). Using this technique, a node wishing to transmit a message listens to the shared bus to see if any other node is transmitting ("Carrier Sense Multiple Access"). If no other node is transmitting, then the node begins to transmit while listening to see if the message is being garbled by another node's transmission ("Collision Detection"). If a collision is detected, the node will wait a while (back off) and then will try to transmit again. If the retry encounters a collision, the whole process is repeated. Thus, the CSMA/CD control technique allows orderly shared access to the bus. A node can transmit a message to all of the other nodes in the network. To support virtual circuits in this broadcast environment, however, a form of control is needed to route messages between specific nodes.

A data encapsulation technique is used to control the routing of messages between nodes. To make nodes distinguishable from each other, every node is assigned a unique number called an address. With this additional technique, a node wishing to transmit a message first encapsulates, or surrounds, the message with control information, forming a new message called a packet. The node then transmits the packet using the CSMA/CD technique. The surrounding control information includes the address of the remote, or destination, node which is to receive the message. The actual routing
is achieved by having nodes ignore all packets they receive except those packets with a
destination address that matches their own address. Since node addresses are unique,
only the specifically addressed node will act upon the broadcast message.

A virtual circuit (or session) is established between two node units when they
agree to send packets addressed to each other. Each node then packages the data
it receives from its attached user device into packets and sends them to the other
node. When a node receives a packet from the other, it unpacks the data from the
packet and transmits the data to its attached user device. Thus, the two devices can
communicate with each other over the virtual circuit. Other techniques, described in
the following chapters, are employed in the establishment and maintenance of a session
in the detection of transmission errors and in the recovery from errors.

Initial setup of an ADSL LAN system would require only the routing of a single
cable near the possible devices to be interconnected. The use of a single cable results in
much lower cable cost and routing complexity compared to that of the point-to-point
interconnect method. To connect a user device to the LAN simply requires the addition
of a node unit to a convenient point on the bus cable. As will be seen in Chapter 3, the
node units can be manufactured inexpensively. The lack of centralized control provides
substantial reliability. If a node fails, messages can still be transmitted over the bus,
provided the failure is not a continuously transmitting node. The decentralized control
also leads to a very maintainable system. The only central component of the system
is the passive bus cable. The active components are the nodes, which are functionally
identical and thus interchangeable.
Chapter 2

A User’s Manual

When a node unit is reset the node runs a series of self tests, sets up various operating attributes, and then enters either Autobaud Mode or Command Mode. In Autobaud Mode the node attempts to set the baud rate of the user interface automatically to match that of the attached user device. Once a baud rate has been determined, the node leaves Autobaud Mode and enters Command Mode. In Command Mode the node accepts commands via the asynchronous RS-232-C serial user interface from the user device to control the node’s operation. It is from the Command Mode that a user can issue a command to establish a session and enter the Connect Mode. In Connect Mode the node transfers data between the user device and a remote user device via the virtual circuit. See Figure 2.1.

2.1 Self Tests of the Hardware

On reset (due either to powering on or to pushing the reset button) a node unit runs a series of tests on itself. The tests check the integrity of the processor, EPROM, RAM, and user interface. Results of the test are indicated on the User Status LED. When
Figure 2.1: Overall Operating Modes of a Node Unit
Table 2.1: User Status LED Display Sequence after Reset

<table>
<thead>
<tr>
<th>Status LED</th>
<th>Hardware Self Tests</th>
<th>Indication</th>
</tr>
</thead>
<tbody>
<tr>
<td>ON</td>
<td>Power on, LED functioning.</td>
<td></td>
</tr>
<tr>
<td>OFF</td>
<td>Processor functioning.</td>
<td></td>
</tr>
<tr>
<td>ON</td>
<td>EPROM functioning.</td>
<td></td>
</tr>
<tr>
<td>OFF</td>
<td>RAM functioning.</td>
<td></td>
</tr>
<tr>
<td>ON</td>
<td>User interface functioning.</td>
<td></td>
</tr>
<tr>
<td>OFF</td>
<td>Enter Autobaud or Command Mode.</td>
<td></td>
</tr>
</tbody>
</table>

power is first applied or when the reset button is depressed, the status LED should be lit. If the processor is running properly, then the status LED should extinguish approximately a-half second later. The node unit then executes sequentially a test of the EPROM, the RAM, and the user interface. After the successful completion of each test, the node flashes the status LED on for approximately a-half second. If a test fails, the status LED is not flashed, any remaining tests are not executed, and the node unit enters a halt loop. Otherwise, after successfully completing the tests, the node will continue with the reset process. Table 2.1 summarizes the reset testing sequence.

2.2 Default Operating Attributes

Each node in a network has a set of attributes which controls the operation of that particular node. Most of these attributes define how the node's user interface operates. Others define the node's unique address, Command Mode operation, etc. After the
successful completion of the self tests, these operating attributes are set to default values. The default values for some of the attributes are specified via settings of DIP switches (See Table 2.2), while the default values for other attributes are fixed. All of the operating attributes can be changed by issuing commands in Command Mode. Details of the attributes and how they affect the operation of the node will be discussed in the remainder of the chapter.

2.3 User Interface

For historical and economical reasons, an asynchronous serial communication interface based on the Electronic Industries Association's (EIA) RS-232-C standard has become widely used by the computer industry. Most terminals, personal computers, and peripheral devices come equipped with this type of communications interface. Because of this widespread use, the asynchronous RS-232-C serial interface was chosen as the user interface on the node units.

Although the mechanical and electrical aspects of the interface have become very standardized in use, the data format and control protocols have not. The specific data format and control protocol of the user interface are defined by several attributes. Through careful configuration of these attributes, practically any device can be attached to a node unit.
Table 2.2: DIP Switch Attribute Settings

<table>
<thead>
<tr>
<th>DIP Switch 1</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Bits</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>Rate</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Baud Rate</td>
</tr>
<tr>
<td>0 0 0</td>
<td>9600</td>
<td>1</td>
<td>Stop Bits</td>
<td></td>
</tr>
<tr>
<td>1 0 0</td>
<td>4800</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 1 0</td>
<td>1800</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 1 0</td>
<td>1200</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 0 1</td>
<td>300</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 0 1</td>
<td>150</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 1 1</td>
<td>110</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 1 1</td>
<td>75</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>bit 4</td>
<td>0</td>
<td>Parity disabled</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Parity enabled</td>
<td></td>
<td></td>
</tr>
<tr>
<td>bit 5</td>
<td>0</td>
<td>Even parity</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Odd parity</td>
<td></td>
<td></td>
</tr>
<tr>
<td>bit 6</td>
<td>0</td>
<td>8-bit word length</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>7-bit word length</td>
<td></td>
<td></td>
</tr>
<tr>
<td>bit 7</td>
<td>0</td>
<td>Flow control enabled</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Flow control disabled</td>
<td></td>
<td></td>
</tr>
<tr>
<td>bit 8</td>
<td>0</td>
<td>XON/XOFF flow control</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>RS-232-C flow control</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DIP Switch 2</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Bits</td>
<td>8</td>
<td>1</td>
<td></td>
<td>Low order byte of node address</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DIP Switch 3</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Bits</td>
<td>4</td>
<td>1</td>
<td></td>
<td>Low nibble of high order byte of node address</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bit 5</td>
<td>0</td>
<td>Verbose Mode enabled</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Verbose Mode disabled</td>
<td></td>
<td></td>
</tr>
<tr>
<td>bit 6</td>
<td>0</td>
<td>Connect Mode escape enabled</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Connect Mode escape disabled</td>
<td></td>
<td></td>
</tr>
<tr>
<td>bit 7</td>
<td>0</td>
<td>DTR control disabled</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>DTR control enabled</td>
<td></td>
<td></td>
</tr>
<tr>
<td>bit 8</td>
<td>0</td>
<td>Autobaud Mode disabled</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Autobaud Mode enabled</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Notes: Switch open = 1 and switch close = 0. The high nibble of the high order byte of the node address is assumed 0000.
Table 2.3: RS-232-C Electrical Specifications

<table>
<thead>
<tr>
<th>RS-232-C Electrical Specifications</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mode of operation</td>
</tr>
<tr>
<td>Logical 1 (Asserted)</td>
</tr>
<tr>
<td>Logical 0 (Dropped)</td>
</tr>
<tr>
<td>Maximum cable length</td>
</tr>
<tr>
<td>Output short circuit</td>
</tr>
</tbody>
</table>

2.3.1 Mechanical and Electrical Aspects

All of the user interface signal lines are routed to a subminiature "D" 25 pin female connector (type DB-25S). The key aspects of the electrical signaling that are used are summarized in Table 2.3. Translation between this RS-232-C electrical signaling and standard TTL electrical signaling can be easily achieved using readily available integrated circuits.

Table 2.4 lists the pin assignments and functions of each signal available on the user interface. All pins not listed are unassigned and not used by the node unit. In many cases, when RS-232-C control functions are not enabled, only three signal lines (pins 2, 3 and 7) are necessary to connect a user device.

2.3.2 Character Format

Figure 2.2 shows the possible asynchronous character formats that can be used on the user interface. The data word length, the type of parity bit, and the number of stop bits actually used depends on the node unit's current attribute values. In the case of
**Table 2.4: User Interface Pin Assignments**

<table>
<thead>
<tr>
<th>Name</th>
<th>Pin</th>
<th>Source</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>TD</td>
<td>2</td>
<td>User</td>
<td>Input data from user device to node unit.</td>
</tr>
<tr>
<td>RD</td>
<td>3</td>
<td>Node</td>
<td>Output data from node unit to user device.</td>
</tr>
<tr>
<td>RTS</td>
<td>4</td>
<td>User</td>
<td>If RS-232 Flow Control is enabled, then this signal should be asserted by the user device to indicate it is ready for data from the node unit. If RS-232 Flow Control is disabled, then this signal is ignored.</td>
</tr>
<tr>
<td>CTS</td>
<td>5</td>
<td>Node</td>
<td>If RS-232 Flow Control is enabled, then this signal is asserted by the node unit to indicate it is ready to accept data from the user device. If RS-232 Flow Control is disabled, then this signal is always asserted by the node unit.</td>
</tr>
<tr>
<td>DSR</td>
<td>6</td>
<td>Node</td>
<td>Always asserted when the node unit is functioning.</td>
</tr>
<tr>
<td>DCD</td>
<td>8</td>
<td>Node</td>
<td>Asserted by the node unit when ever a session is active between the local node and some remote node. Dropped when no session is present.</td>
</tr>
<tr>
<td>DTR</td>
<td>20</td>
<td>User</td>
<td>If DTR Control is enabled, this signal must be asserted by the user device to allow session establishment and to maintain a session. If this signal is dropped during an active session, the the session is terminated. If DTR Control is disabled, then this signal is ignored.</td>
</tr>
<tr>
<td>TEST</td>
<td>25</td>
<td>User</td>
<td>Test signal input. This signal is used by the node unit during external loopback tests. It serves no interface function.</td>
</tr>
<tr>
<td>FG</td>
<td>1</td>
<td></td>
<td>Frame Ground. Connected to the node unit’s signal ground. May be isolated from signal ground if necessary (see jumpers E14 and E15).</td>
</tr>
<tr>
<td>SG</td>
<td>7</td>
<td></td>
<td>Signal Ground.</td>
</tr>
</tbody>
</table>
word length and parity type, there is a corresponding attribute which specifies what is used. The number of stop bits used, however, depends on the current baud rate in use (see Table 2.2). When a parity bit (even or odd) is used, the node unit generates the correct value on all transmissions to the user device. The node unit, however, ignores the parity bit on characters coming from the user device.

2.3.3 Bit Rate

The user interface can operate over a range of standard baud (bit) rates: from 75 bits/second to 9600 bits/second. If Autobaud Mode is not enabled, the default baud rate is determined at reset by a DIP switch setting (see Table 2.2). The baud rate attribute may be changed via the Command Mode.

Autobaud Mode provides a powerful method for setting the operating baud rate. In Autobaud Mode a node unit attempts to determine at what baud rate the user device is operating. The user device sends one or more ASCII carriage return characters
(0D<sub>HEX</sub>) in rapid succession to the node unit until the node ascertains the baud rate. Upon successfully determining the baud rate, the node unit enters Command Mode and the Command Mode sign-on message is displayed.

If enabled, a node unit will enter Autobaud Mode after reset and after the termination of a session. Autobaud Mode will also be entered when enabled from Command Mode. A node will leave Autobaud Mode when either the user device's baud rate is determined or when a session is established by a remote node unit. In the latter case the node unit enters Connect Mode and will operate the user interface based on the current baud rate attribute.

2.3.4 Flow Control

During a session in Connect Mode, node units support the capability to control the flow of data to and from the attached user devices. If data is coming from the user interface too fast for the user device to process, the user device may request the node to stop sending data. There are several reasons why a node unit may need to request the attached user device to stop sending data:

- The device attached to the remote node invoked flow control.
- The remote device is operating at a slower baud rate than the local device.
- Network congestion may delay the node unit from transmitting the user data to the remote node.
Two methods of flow control are available on the user interface: RS-232-C control signal hand shaking and XON/XOFF character hand shaking. If RS-232 flow control is enabled, the flow of data is controlled by the RTS and CTS hand shaking signals on the user interface connector (see Table 2.4). If XON/XOFF flow control is enabled, data flow is controlled by the use of two characters: XON (an ASCII control Q, 11_{HEX}) and XOFF (an ASCII control S, 13_{HEX}). When either the node unit or the user device needs to invoke flow control (i.e., stop the data transmission to itself), it sends an XOFF character. The flow control is removed when an XON is sent. Note that flow control can be invoked in either direction: one direction by the node unit and the other by the user device. To prevent a deadlock of the user interface, the node unit will transmit a XON/XOFF as needed even if the user device has invoked flow control.

2.4 Command Mode

When a node unit is in Command Mode the attached user device can establish a session, change operating attributes, and query the node's status by issuing commands on the user interface. When a node enters the Command Mode either from reset or from Autobaud Mode, a sign-on message is displayed. The sign-on message includes the node's software version number and the node's current network address. After the sign-on message is sent, the node unit prompts for a command by sending an ASCII colon (3A_{HEX}). It is at this point that commands can be issued by the user device. Figure 2.3 shows a sample sign-on message and prompt.

The following sections describe the syntax and details of the commands. A note of
caution: many of the commands change the operating attributes of the user interface. Once one of these commands (e.g., Baud) is issued, the user device must change its communication interface to match the new setting before communication is possible again with the node.

2.4.1 Syntax

The format of command is

```
<command name> <parameter name><cr>
```

where the command name and possible parameter name are separated by one or more spaces. The command is then terminated with an ASCII carriage return. Only the characters that uniquely distinguish the command or parameter name need be typed. An ASCII backspace (08FE) or an ASCII delete (7FHE) may be used at any time to correct any entry errors.

A node unit can operate in a conversational mode which is helpful when a person is issuing commands directly. If Verbose Mode is enabled, a node unit provides name completion, interactive help, and extended status and error messages. When
the characters that uniquely distinguish the command or parameter name (or more characters of the name) have been entered, a space or carriage return causes the node to complete the name. An ASCII question mark (3F_{HEX}) can be entered at any point causing the display of appropriate help information. Figure 2.4 is an example of the interactive help. The effect of Verbose Mode on status and error messages is discussed in Section 2.4.3.

2.4.2 Command Detail

The following notational conventions are used to describe command format in this section:

- **Uppercase**: Characters shown in uppercase must be entered to uniquely define the command name or parameter name.

- **Lowercase**: The remaining lowercase characters of the command name or parameter name are not significant and may be omitted.

- **{ }**: One and only one of the enclosed parameters must be entered. The braces are not entered.

- **|**: The vertical bar separates options within braces { }.

- **xxxx**: A four-digit hexadecimal number. Leading zero (0) digits may be omitted.

- **xx**: A two-digit hexadecimal number. Leading zero (0) digits may be omitted.
Figure 2.5 is a summary of the available commands. Format and details of each command follow.

ADDRESS Command

Format: \textit{Address} \{ xxxx \}

Sets the node unit’s address to the specified parameter. If issued when a session is active the command has no effect. Default value is specified by DIP switch settings.

AUTOBAUD Command

Format: \textit{Auto baud} \{ Enable | Disable \}

Enables or disables Auto baud Mode according to the specified parameter. If Auto baud Mode is enabled when no session is active, Command Mode is entered. Default value is specified by DIP switch settings.

BAUD Command

Format: \textit{Baud} \{ 75 | 110 | 150 | 300 | 1200 | 1800 | 4800 | 9600 \}

Sets the user interface baud rate to the specified parameter. Default value is specified by DIP switch settings.

CONNECT Command

Format: \textit{Connect} \{ xzzz \}

Requests a session with a remote node unit with an address specified by the parameter. If a session is established, Connect Mode is entered. If a session is already
• Node Attribute Control Commands:
  - ADDress { xxxx }
  - AUTobaud { Enable | Disable }
  - Baud { 75 | 110 | 150 | 300 | 1200 | 1800 | 4800 | 9600 }
  - DTr { Enable | Disable }
  - Escape { Disable | xxxx }
  - Flow { Xon/xoff | Rs232 | Disable }
  - Parity { Odd | Even | Disable }
  - Timeout { xxx }
  - Verbose { Enable | Disable }
  - Word { 8bit | 7bit }

• Session Control Commands:
  - Connect { xxxx }
  - Disconnect
  - Resume

• Status Report Commands:
  - Help
  - Status

Figure 2.5: Command Summary
active, the command has no effect.

**DISCONNECT Command**

**Format:** *DISconnect*

Requests the currently active session be terminated. If issued when no session is active, the command has no effect.

**DTR Command**

**Format:** *DTr { Enable | Disable }*

Enables or disables DTR control on the user interface according to the specified parameter (see Table 2.4 on page 14). Default value is specified by DIP switch settings.

**ESCAPE Command**

**Format:** *Escape { Disable | zxxx }*

Sets the Connect Mode escape sequence to a two ASCII character sequence or disables escape according to the specified parameter. Default value of "enabled" or of "disabled" is specified by DIP switch settings. If enabled, the default Connect Mode escape sequence is \( 18_{16} \) (control X, control X).

**FLOW Command**

**Format:** *Flow { Xon/xoff | Rs232 | Disable }*
Enables either RS-232 or XON/XOFF flow control or disables flow control on the user interface according to the specified parameter. Default value is specified by DIP switch settings.

**HELP Command**

Format: `Help`

Displays a summary of available commands.

**PARITY Command**

Format: `Parity { Odd | Even | Disable }`

Sets the parity on the user interface to even or odd parity, or disables parity according to the specified parameter. Default value is specified by DIP switch settings.

**RESUME Command**

Format: `Resume`

Re-enters Connect Mode after an escape from Connect Mode to Command Mode. If issued when no session is active the command has no effect.

**STATUS Command**

Format: `Status`

Displays the current attribute values of the node unit. The node’s local address, user interface attributes, and current session status are displayed. Figure 2.6 is an example of the status display of a node unit with an active session.
:status

Node Status –
   Local Node Address - 0123
   Autobaud Mode - Enabled
   Baud Rate - 300
   DTR Control - Disabled
   Connect Escape - Enabled, 1818
   Flow Control - XON/XOFF
   Parity - Disabled
   Buffer Timeout - 05
   Verbose Mode - Enabled
   Word Length - 8 bit
   Current Session - Active with remote node OABC

Figure 2.6: Sample Status Command Display

TIMEOUT Command

Format: Timeout {

Sets the user interface character buffering timer restart value to the specified parameter. The value is specified in units of 12.8 milliseconds. Default value is 05_HEx (64 milliseconds).

VERBOSE Command

Format: Verbose { Enable | Disable }

Enables or disables Verbose Mode according to the specified parameter. Default value is specified by DIP switch settings.
WORD Command

Format:  \textit{Word \{ 8bit | 7bit \}}

Sets the user interface word length according to the specified parameter. Default value is specified by DIP switch settings.

2.4.3 Status and Error Messages

A node unit will send unsolicited messages to the user device in response to some status changes or error conditions. The format for these status/error messages is

\textit{!<bell>zzzyy - Verbose Message.}

Messages begin with an ASCII exclamation point (21_{HEX}) followed by an ASCII bell character (07_{HEX}). Three characters, \textit{zzz}, follow, identifying the software module that issued the message: \textit{NET} for the Network Layer, \textit{DLK} for the Data Link Layer, and \textit{COM} for the Command Layer. Two more characters, \textit{YY}, follow, which uniquely identify the particular status/error condition being reported. If Verbose Mode is enabled, a text message follows describing the status/error condition. The following are the details of possible status/error messages.

\textit{!COMCE - Command Syntax Error.}

There was a syntax error in the command name issued by the user device.

\textit{!COMPE - Parameter Syntax Error.}

There was a syntax error in the command name issued by the user device.
!COMSL - Command String Too Long.

The command string (command name, parameter name and spaces) issued by the user device was too long.

!NET01 - Retry exceeded, No response from remote.

The maximum number of retransmissions of a packet has been exceeded. The current session, if any, is ungracefully terminated.

!NET02 - Watch dog time out, No response from remote.

The maximum number of retransmissions of a watch dog packet has been exceeded. The current session is ungracefully terminated.

!NET11 - Session established via local.

A session, requested by the local user device, has been successfully established with the remote node unit.

!NET12 - Remote is busy.

The remote node unit currently has a session with another node. The requested session is not established.

!NET13 - Remote is not available.

The remote node unit is currently unable to accept a session. The requested session is not established.
!NET14 - Session closed via local.

The current session has been gracefully terminated by a request from the local user device.

!DLK01 - Backoff retry exceeded.

The maximum number of backoff retransmissions of a packet has been exceeded. The current session is ungracefully terminated.

!DLK02 - Network jammed.

The maximum number of access attempts has been exceeded. The current session is ungracefully terminated.

!DLK03 - Defer timeout.

The maximum number of access deferrals has been exceeded. The current session is ungracefully terminated.

### 2.5 Connect Mode

Figure 2.7 is an example of a session request via the Connect command which was successfully established. At this point both nodes, local and remote, enter Connect Mode. Any data that the attached user device sends to the user interface is now relayed by the local node to the remote node. The remote node then sends the data.
to its attached user device. Data is similarly sent from the remote user device back to the local device. Thus a virtual circuit exists between the two devices.

2.5.1 Session Establishment

A node can have a session established in two ways: by a command request issued from the locally attached user device or by a remote user request. In either case, if DTR control is enabled, the DTR signal on the user interface must be asserted by the local user device before the session can be established. The DTR signal can give the user device control over whether a session is established or not.

When a session is established, the node unit will assert the DCD signal on the user interface (see Table 2.4 on page 14) and light the user status LED to indicate that a session is active. During the course of the session the user status LED will show activity on the virtual circuit by flashing the LED off momentarily whenever a packet is sent or a received packet is acknowledged.
2.5.2 The Virtual Circuit vs. the Physical Circuit

The virtual circuit provided by the ADSL LAN differs from a physical point-to-point connection in several important ways. First, with the virtual circuit, there is a full translation of communication protocol between the two attached user devices. The user of one node may be operating with one set of attributes while the user of the other node is operating with a totally different set of attributes. Thus, for example, one user device operating with XON/XOFF flow control and a 9600 baud rate can reliably communicate with another device operating with RS-232 flow control and a 300 baud rate. Such a translation is not possible with the simple cable connection.

The flexibility and versatility provided by the LAN is not, however, without some costs. Namely, data transmission through a virtual circuit suffers delays. Delays arise from the data packaging process, from contending for access to the shared bus, and from the transmission time on the bus. By using a bus transmission rate faster than that of the user interface, bus transmission delays are minimized. The number of contention delays can be reduced by sending fewer packets, each with more data. Larger packets also make better utilization of the bus bandwidth. As the packet size increases, the overhead ratio of encapsulating control information to actual user data sent decreases.

To limit the packaging delay while maintaining large packets, a buffering timer is used. Whenever the node unit receives a character from the user device, the node places the character in a buffer and resets the timer. While the node waits for another character from the user, the timer counts down. If the timer reaches zero before another character is received, all the data in the buffer is packaged up and sent over to
the remote node. However, if a character is received before the timer reaches zero, the character is added to the buffer and the timer is reset. Thus, as long as the user device sends characters at a rate faster than the timer restart value, the buffer of data will continue to grow and not be sent. If the buffer reaches a maximum size (122 bytes), it is packaged and sent regardless of the timer's value. The timer's restart value is a node attribute, which can be changed via Command Mode. The default restart value is 64 milliseconds.

Another important difference between the virtual circuit and the physical circuit is data transparency. In the physical circuit characters pass between the two devices unchanged. With the virtual circuit, on the other hand, there are two cases where a character sent by a device does not reach the other device. In the first case, if XON/XOFF flow control is used on a user interface, XON and XOFF characters sent by the user device are absorbed by the node unit and are not sent to the remote device. This will not be a problem in most applications if the proper type of flow control is set up. The second case occurs when the Connect Mode escape sequence is enabled. If this escape sequence is enabled, the node will exit Connect Mode when the attached user device sends a special two-character sequence. This character sequence is absorbed by the node unit. Problems can arise when the data stream from the user device contains the escape sequence. By either changing or disabling the Connect Mode escape sequence, the trouble can be avoided.
2.5.3 Session Termination

A session between two node units can terminate either gracefully or ungracefully. When both nodes in the session agree to terminate, the session is gracefully terminated. When a session is terminated, either gracefully or ungracefully, the node unit will enter either Command Mode or Autobaud Mode depending on whether Autobaud Mode is disabled or not. As in the session establishment, either node can request that the session be terminated. If DTR control is enabled and the DTR signal is dropped, the session will terminate gracefully. Another way to terminate a session gracefully is to utilize the Connect Mode escape sequence. If enabled, the node unit will leave Connect Mode and enter Command Mode when the user device sends a specific two character sequence (e.g., control X, control X). Note that although the node is now in Command Mode the session is still active and Connect Mode may be re-entered using the Resume command. In Command Mode, a Disconnect command can be issued which will gracefully terminate the session.

Ungraceful session termination arises when one node terminates a session and the other node is not informed. This can happen when an unrecoverable node error occurs or a node is reset. The uninformed node will continue to assume the session is active until it transmits a packet. At this time an unrecoverable error will occur, since the packet is not acknowledged by the missing node. During periods of no activity on a virtual circuit, the node units will exchange “watch dog” packets with each other approximately every 30 seconds. This guarantees that the uninformed node in an ungraceful termination will eventually learn of the termination.
Chapter 3

The Hardware and Software

3.1 Hardware

A block diagram of the node unit hardware is shown in Figure 3.1. The hardware is microprocessor based. All components are readily available and inexpensive. Appendix A contains a component list and detailed schematics of the circuitry. Contact the Advanced Digital Systems Laboratory for the most recent revisions.

3.1.1 Microprocessor

Central to the node unit is the 8032 single-chip microprocessor manufactured by Intel Corporation. The 8032 processor contains a control-oriented 8-bit CPU; 256 bytes of read/write data memory; 32 I/O lines; three 16-bit timer/counters; a six-source, two-priority level, nested interrupt structure; a programmable serial I/O port; and an on-chip oscillator with clock circuitry. The 8032 processor can address up to 64k bytes of external data memory and up to 64k bytes of external program memory. For further
User Interface
- Based on INS8250 ACE
- Async RS-232C
- 110 - 9600 Baud

User Equipment

Program Storage
- Based on 2764/128 EPROM
- 8k/16k bytes

CPU
- Based on Intel 8032
- 8-bit CPU
- 256 bytes RAM
- Three 16-bit timers
- Async Serial Port

Data Storage
- Based on HM6116/HM6264 CMOS RAM
- 2k/8k bytes

Switch Bank
- Three 8-switch DIPs with scanning logic

Network Transceiver
- Based on 75176
- RS-485

Twisted-Pair Bus

Figure 3.1: Block Diagram of the Node Unit Hardware
details on the 8032 processor's architecture and instruction set see the Microcontroller Handbook [1].

A 12.00 MHz crystal provides the time reference for the oscillator and clock circuitry. A simple resistor/capacitor network is used to generate a power-on reset signal. An octal D-type latch, the 74LS373, is used to separate the multiplexed address/data bus of the 8032 processor. This provides separate address and data signals for the external memory and user interface circuitry. The user status LED is controlled by an I/O pin of the 8032 processor via an inverting buffer.

The on-chip serial port is used as the communications channel to the shared cable bus of the network. The ADSL LAN software programs one of the timer/counters to generate timing for the serial port. Another timer/counter is used for software timing. The third timer/counter is not currently utilized by the software. The external interrupt INTI pin is tied low. This allows an interrupt to be generated under software control.

The EA pin, which controls instruction fetches, can be jumpered high or low. Normally it is jumpered low (jumpers E5 and E6 connected), forcing the processor to execute from external program memory. If jumpered high (jumpers E4 and E5 connected), an 8052, 8051 or 8751 processor or an ICE51/52 emulator can replace the 8032 processor in the hardware. These are all devices with internal program memory. This replacement can be helpful for testing purposes. However, note that the current version of the ADSL LAN software requires the use of an 8032 processor. The ADSL LAN software utilizes the addition on-chip RAM present in the 8032 processor.
3.1.2 External Memory

An ultraviolet-erasable, programmable ROM (EPROM) device is used for the external program memory. Either a 2764 or a 27128 industry standard type EPROM can be used in the hardware without modification. The 2764 is an 8k by 8-bit device and the 27128 is a 16k by 8-bit device. However, note that the current version of the ADSL LAN software requires the use of a 27128 type EPROM. For compatibility with the 8032 processor's access timing, an EPROM with an address-to-output delay of 250ns or less must be used. If an 8751 processor is used with the external program memory present (as is done with the hardware diagnostic software), an EPROM with an address-to-output delay of 200ns or less must be used.

A static CMOS RAM device is used for the external data memory. Either of the HM6116 or HM6264 memory devices manufactured by Hitachi can be used. The HM6116 is a 24-pin 2k by 8-bit device and the HM6264 is a 28-pin 8k by 8-bit device. If an HM6264 is used, jumpers E1 and E2 must be connected. If an HM6116 is used, pin 1 of the device must be installed into pin 3 of the socket and jumpers E2 and E3 must be connected. The current version of the ADSL LAN software requires only 2k bytes of external data memory. Therefore, either of the two devices can be used. A device with an access time of 200ns or less is compatible with the processor's external data memory timing. Pin compatible CMOS RAM devices are available from other manufacturers.

The base address of the RAM is at address 0000_{HEX} in the external data memory space. Note that there is no conflict between the external data memory and the external
program memory, since the two reside in two different address spaces. Each space has a different set of control signals from the processor: \texttt{PSEN} for the program memory, \texttt{RD} and \texttt{WR} for the data memory.

### 3.1.3 Switch Bank

A bank of three 8-switch DIP switch packages, a total of 24 bits of information, is available on the node unit to be read by the processor. The functional definition of each switch is entirely determined by the software. The three lower bits of Port 1 of the processor supply a scan address to three 3-to-8 open collector decoders. Each decoder's outputs are then used to select corresponding switches on a DIP package. Each DIP switch package has the other side of its switches all tied together and routed into the processor. See Figure A.4 of Appendix A. Thus the processor can read the setting of any switch by selecting the appropriate address and reading the appropriate DIP package input signal.

### 3.1.4 User Port

The user interface hardware is based on the INS8250A Asynchronous Communications Element (ACE) manufactured by National Semiconductor Corporation. The ACE performs serial-to-parallel conversion on the data characters received from the user device and parallel-to-serial conversion on the data characters received from the processor. The serial interface characteristics (data word length, parity bit generation and detection, stop bit generation, and baud rate generation) are fully programmable.
by the processor. The ACE includes a fully prioritized interrupt system and internal diagnostic capabilities. For more detailed information see National Semiconductor's data sheet [3].

Although INS8250A ACE includes on-chip oscillator circuitry, this circuitry was not used because it would not oscillate reliably. Instead, an external oscillator circuit made up of TTL gates and a 1.8432MHz crystal is used. Depending on the characteristics of the particular crystal used, a small trimming capacitor across the crystal may be necessary. The availability of several chip enable lines on the ACE makes the decode logic for chip selection simple. The ACE's control and data registers are memory mapped into the external data memory space. The base address is mapped to $8000_{HEX}$.

The serial input and output lines of the INS8250A ACE are interfaced to the user connector via a 75189 line receiver and a 75188 line driver, respectively. These two devices provide full conformance with the electrical specifications of the EIA RS-232-C standard [5]. The 75189 and 75188 devices are manufactured by Texas Instruments, but interchangeable devices are also available from other manufacturers.

### 3.1.5 Transceiver

The transceiver circuitry interfaces the on-chip serial port of the processor to the shared twisted-pair cable bus. The circuitry consists of a differential line driver and a differential line receiver attached directly to the twisted pair. A single transceiver integrated circuit (the 75176) or separate driver (the 75174) and receiver (the 75175) integrated circuits can be used. The appropriate jumper connections must be made (see
Appendix A). For either case the driver enable control is operated by an output of the processor. The jumpers also allow the circuitry to be configured to operate with transceiver logic mounted off the node unit itself. This option separates the transmit, receive, and driver enable signals on the edge connector. This remote transceiver option should be used when the node unit itself can not be physically near the cable bus. For more information on the electrical characteristics of the line drivers and line receivers see references [4,5] and Chapter 4.

### 3.1.6 Hardware Diagnostic Software

During the development of the hardware and controlling software for the ADSL LAN, a hardware diagnostic software package was written. The software is designed to help find and diagnose problems with the node unit hardware by replacing the normal controlling software in the node. Only a minimal amount of the hardware needs to be functioning for the diagnostic software to run. Appendix B contains a listing of the hardware diagnostic software. Contact the Advanced Digital Systems Laboratory for the most recent revisions.

There are four different versions of the diagnostic software corresponding to four processor/memory environments. By setting an appropriate compiler switch in the source code, a version can be generated to run in an 8751 processor, an ICE51 emulator, an 8032 processor and a 2764 EPROM, or an 8032 processor and a 27128 EPROM. The versions that operate in a smaller memory environment contain only a basic set of test routines, while the larger versions contain the full complement of tests. The
8751 processor and ICE51 emulator versions are particularly useful because they do not depend on the external memory circuitry.

On reset of the diagnostic software the user status LED should be initially lit, then extinguished if the processor is running properly. A test of the user interface is then run. If the test is successfully completed, then the status LED is flashed on for approximately a-half second. After completing the user interface test, the user interface is set to a 9600 baud rate, no parity bit, and 8-bit word length regardless of the current DIP switch settings. A sign-on message and prompt are then displayed. At this point commands can be issued to execute a variety of tests. Figure 3.2 shows the commands available with the diagnostic software.

Dn Disable test n.
En Enable test n.
F Run all enabled tests forever.
H Display command help menu.
P Toggle External EPROM size.
Q Toggle quiet mode on/off.
R Toggle External RAM size.
S Show status and availability of all tests.
T Run all enabled tests.
Tn Run test n.

Figure 3.2: Command Summary of the Hardware Diagnostic Software
3.2 Software

Since the node unit hardware is very general in nature, most of the node's functions are implemented in software. Some early software attempts were written in assembly language. As the required complexity of the software became apparent, the use of assembly language for the task looked nearly impossible. A decision was made to use PL/M-51, an implementation of Intel's high-level language compiler for the 8051 processor family. The use of PL/M-51 has allowed faster software development with fewer errors as compared to the use of assembly language. The traditional costs of using a high-level language, namely increased code size and decreased speed, are not too severe with PL/M-51.

3.2.1 PL/M-51 Language

Similar to IBM's PL/1 mainframe language, the PL/M-51 language provides block structure and powerful control constructs for structured programming, data structure facilities, and data type checking. Yet, all of the low-level processor features are still readily accessible. The PL/M-51 compiler translates the language into machine executable object modules. Separate modules can then be linked into one program module using the RL-51 linker utility. For detailed information on the language and compiler see the PL/M-51 User's Guide [2].

The high-level language has been efficiently designed to map into the machine level architecture. For example, the compiler optimizes the use of on-chip RAM by overlaying
local variables. Under this optimization, local variables of different procedures share the same location of on-chip RAM, provided the procedures do not call each other. This makes very efficient use of the limited on-chip RAM. However, it can cause trouble.

The compiler views interrupt service procedures as being called by no other procedures, causing the service procedure’s local variables to overlay other local variables. So when an interrupt occurs (at most any time) the interrupt service procedure can destroy the interrupted procedure’s local variables. Interrupt service procedures should be viewed instead as callable by all procedures, since an interrupt can occur any time. To avoid this trouble, all interrupt service procedures and procedures they call should be declared with the INDIRECTLY-CALLABLE attribute.
Chapter 4

The ADSL LAN Architecture

To make the ADSL LAN architecture more manageable, the system is organized in layers. Figure 4.1 shows how the communication functions are partitioned into a hierarchical set of layers. These layers correspond approximately to the lower layers of the International Standards Organization's Open Systems Interconnection model [6]. Each layer performs a related subset of the functions required in the communication process. A layer relies on the next lower layer to perform more primitive functions while concealing the details of those functions. A layer also provides services to the next higher layer. The Physical Layer and parts of the User Layer are implemented in hardware. The remaining layers are implemented in software that runs on the node hardware. Appendices C through I contain software listings of the modules that comprise the ADSL LAN's controlling software. Each module corresponds to a layer of the architectural model, with the exception of the Timer Server module. This module provides timing services for the entire system.

When a node unit is in Connect Mode, data from the user device is received by the User Layer. The Transport Layer then gathers individual characters from the User
<table>
<thead>
<tr>
<th>Layer</th>
<th>Functions</th>
</tr>
</thead>
<tbody>
<tr>
<td>User Layer</td>
<td>• Interrupt Driven</td>
</tr>
<tr>
<td></td>
<td>• User Flow Control</td>
</tr>
<tr>
<td></td>
<td>• Data Buffering</td>
</tr>
<tr>
<td>Transport Layer</td>
<td>• Session Establishment</td>
</tr>
<tr>
<td></td>
<td>• Session Deactivation</td>
</tr>
<tr>
<td></td>
<td>• Data Blocking/Deblocking</td>
</tr>
<tr>
<td>Network Layer</td>
<td>• Network Flow Control</td>
</tr>
<tr>
<td></td>
<td>• Data Block Buffering</td>
</tr>
<tr>
<td></td>
<td>• Data Block Encapsulation</td>
</tr>
<tr>
<td></td>
<td>• Error Recovery</td>
</tr>
<tr>
<td></td>
<td>• Session Maintenance</td>
</tr>
<tr>
<td>Data Link Layer</td>
<td>• Interrupt Driven</td>
</tr>
<tr>
<td></td>
<td>• Raw Packet Transmission and Reception</td>
</tr>
<tr>
<td></td>
<td>• Error Detection</td>
</tr>
<tr>
<td></td>
<td>• Medium Access Control</td>
</tr>
<tr>
<td></td>
<td>• Medium Contention Resolution</td>
</tr>
<tr>
<td>Physical Layer</td>
<td></td>
</tr>
</tbody>
</table>

Figure 4.1: Overall Layered Structure of the ADSL LAN
Layer into a block and passes the block of characters to the Network Layer. The Network Layer encapsulates the data block to form a packet and passes this packet to the Data Link Layer. The Data Link Layer then transmits the packet over the shared cable bus (the Physical Layer). At the remote node the process is reversed. The Data Link Layer receives a packet from the Physical Layer. The data passes back up through the layers to be eventually sent character-by-character to the attached user device by the User Layer. Thus, a virtual circuit involves processing the local user's data through a series of layers, to a layer common to all nodes (the Physical Layer) and then processing the data in reverse order on the remote node to end up at the remote user device.

When a node unit is in Command Mode, data passes from the user device through the User Layer to the Command Layer. Similarly, data from the Command Layer passes through the User Layer to the attached user device. If a session is currently active while in Command Mode, all data packets received by the node are held in the Network Layer. When the node returns to Connect Mode, the held data is then passed as usual to the user device.

4.1 Physical Layer

The Physical Layer handles bit level character transmission and reception on the shared twisted-pair cable bus. It also covers the electrical signalling aspects used on the bus. All of the Physical Layer's functions are implemented in hardware utilizing the 8032 processor's on-chip serial port and the RS-485 differential transceiver circuitry.
4.1.1 Character Format

The 8032 processor's programmable serial port is operated in Mode 1: 8-bit UART with variable baud rate. Figure 4.2 shows the character format used by the serial port. The 1-to-0 transition of the start bit synchronizes the serial port receiver to the data bit stream that follows. The serial port employs a bit detector for noise rejection. The detector samples the signal values three times over the course of a bit time. The bit's value is then taken to be the value of at least two of the three samples. Note that the bit detector is used on the start bit as well as the data bits. This helps to prevent false triggering of the whole character reception process and reduces data reception errors.

4.1.2 Bit Timing

When operating in Mode 1 the serial port derives its bit timing from an on-chip timer's overflow rate. The timer is configured as an 8-bit counter with automatic reload. With a reload value of $FF_{HEX}$ and the SMOD control bit set, the serial port operates with a bit rate of 62.5k bits per second. The serial port hardware is capable of operating at faster rate, but the node software cannot keep up character handling with rates much

---

Figure 4.2: Physical Layer Character Format
4.1.3 Electrical Aspects

The serial port's input and output are connected to transceiver logic that drives the shared twisted-pair cable bus. The electrical aspects of the transceiver logic is based on the EIA RS-485 standard which was introduced in 1983 as an upgraded version of the EIA RS-422-A standard [4]. Table 4.1 summarizes key electrical aspects of the RS-485 standard. The standard allows multiple drivers and receivers on a single twisted-pair cable.

Together the differential drivers and receivers along with twisted-pair cable make a very low noise transmission medium. Due to mutual coupling of adjacent wires in twisted-pair cable, both wires are equally affected by electromagnetically and electrostatically coupled noise. This results in a net common-mode noise voltage with respect to ground. The common-mode noise voltage is rejected by the receiver, since

Table 4.1: RS-485 Electrical Specifications

<table>
<thead>
<tr>
<th>Mode of operation</th>
<th>Differential</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logical 1 (Asserted)</td>
<td>Terminal A negative with respect to Terminal B</td>
</tr>
<tr>
<td>Logical 0 (Dropped)</td>
<td>Terminal A positive with respect to Terminal B</td>
</tr>
<tr>
<td>Maximum cable length</td>
<td>4000ft</td>
</tr>
</tbody>
</table>

above 62.5k bits per second. The inter-character timing is a function of the Data Link Layer.
the receivers are only sensitive to differential-mode voltages. To further reduce noise problems, the stub (or tap) lengths off the main cable to the transceiver logic should be kept as short as possible. Also, the cable should be terminated at both ends with the cable's characteristic impedance.

4.2 Data Link Layer

Using the basic character transmission and reception services available from the Physical Layer, the Data Link Layer provides packet transmission, reception, and acknowledgement services to the Network Layer. The Data Link Layer also handles transmission error detection, limited error recovery, and shared access to the cable bus using the CSMA/CD technique.

To ensure that packets are received by the nodes they are destined for, each packet broadcast is acknowledged by the destination node when the packet is received correctly. The acknowledgement is a single character sent by the receiver immediately after the packet is received. The Data Link Layer will broadcast a packet for the Network Layer and will return to the Network Layer the acknowledgement character if received. Similarly, the Data Link Layer will give the Network Layer any error-free packets received and will transmit an acknowledgement character for each packet if requested by the Network Layer. Note that the contents of acknowledgements, as well as that of packets, are of no concern to the Data Link Layer.¹ The Data Link Layer

¹This is not quite true. The data length field of the packet as defined by the Network Layer is used by the Data Link Layer to determine the total packet length during packet transmission and reception.
provides only the packet/acknowledgement transaction service. It is up to the Network Layer to evaluate acknowledgements and packets received, and to determine if any acknowledgement should be sent.

4.2.1 Packet Timing

The Data Link Layer imposes the packet construction on the simple character transmission and reception services of the Physical Layer. The Data Link Layer considers a packet to be a series of characters transmitted sufficiently close together. Figure 4.3 illustrates the relative inter-character timing within a packet. Ideally, characters should be transmitted as close as possible to utilize the bus bandwidth efficiently. To allow the character receiver logic of the serial port to reset properly, the absolute minimum inter-character delay must be one bit time. In practice, due to delays of the software in handling the characters, the actual inter-character delay is on the order of 60 microseconds (about 4 bit times). Thus, using the terminology of the CSMA/CD technique, a carrier is sensed (or detected) whenever characters are received within an inter-character delay of each other.

A timer is used to implement this carrier detection feature. Whenever a character is received from the Physical Layer, the timer is reset. While the Data Link Layer
waits for another character, the timer counts down. If the carrier timer reaches zero, the carrier is considered lost. Thus, the restart value used to reset the timer should be at least equal to the maximum inter-character delay. This will avoid losing the carrier while receiving a packet. The actual restart value used is determined by the literal \(^2\) `carrier_timer_restart` declared in the Data Link Layer software module (see Appendix H).

Figure 4.4 illustrates the timing relationships between a packet transmission, its acknowledgement character, and the next packet transmission. After a node transmits a packet, there is a small period of time, or window, in which the receiving node can acknowledge the packet. Normally the node which transmitted the packet will receive a character within this window and will pass it to the Network Layer. If the node does not receive a character within this window, the packet is considered not acknowledged and the event is reported to the Network Layer.

The acknowledgement window should be long enough to allow a receiving node’s Network Layer to evaluate the received packet and possibly respond with an acknowledgement. This response time is dependent on the software. Typical response times

\(^2\)This is a PL/M-51 language literal.
of 450 microseconds (about 28 bit times) have been measured. The window timing is implemented with a count down timer. The timer's restart value (the window's width) is determined by the literal \texttt{ack.timer.restart} declared in the Data Link Layer software module (see Appendix H). Notice that the \texttt{carrier.timer.restart} literal must have a value larger than the \texttt{ack.timer.restart} literal value to avoid loss of carrier in the acknowledgement window.

4.2.2 Error Detection

To detect transmission errors in a packet, the Data Link Layer appends a checksum character to the packet received from the Network Layer (see Figure 4.5). As the Data Link Layer transmits the Network Layer's packet, a sum is computed of all the characters transmitted. This sum is then transmitted as the last character of the packet. As the Data Link Layer receives a packet, a sum of the received characters is computed. This sum is then compared with the last character received (the checksum transmitted with the packet). Normally, if no transmission errors have occurred, this newly computed checksum is equal to the transmitted checksum. However, if a transmission error occurs (i.e., the node does not receive the same character values transmitted) then the two checksums would not be equal.\footnote{There is a small probability that the error in the packet is such that the two checksums are equal. Hence, the error would not be detected. There are more rigorous error check methods (e.g., cyclic redundancy checks); however, they require much more computational overhead.}

Whenever a packet is received correctly (no error detected), the Data Link Layer
passes it to the Network Layer for evaluation. Depending on the results of the evaluation, the Network Layer may request the Data Link Layer to send an acknowledgement. Whenever an error is detected in a received packet it is discarded and not passed to the Network Layer. Notice that this means that an incorrectly received packet will never be acknowledged.

4.2.3 Collision Avoidance, Detection and Recovery

Since the transmission medium of the ADSL LAN is shared among all the nodes, the nodes must take turns transmitting on the cable bus to avoid transmitting simultaneously (colliding). As a first step in avoiding collisions, the Data Link Layer will not begin a packet transmission while a carrier is detected. Thus, the Data Link Layer will not begin transmitting if another node is currently transmitting. However, due to limitations in the carrier detection method and signal propagation delays on the cable bus, collisions can still occur. When a node begins a packet transmission, it takes a finite period of time for the signal of the first character to propagate to all the nodes on the cable bus. Additionally and more importantly, a node must receive the entire first character before the node detects a carrier. Thus, there is a period of time at the start
of a packet transmission that another node, wishing to transmit also, will not detect a carrier yet and will begin transmitting. This period of time, or collision window, is approximately equal to a character transmission time, since the transmission time is much larger than propagation delays.

To reduce this possibility of collision, nodes do not immediately begin a packet transmission when they find the cable bus not in use (i.e., no carrier detected). Instead, when a node wishing to transmit finds no carrier, the node first waits (defers) a random period, then checks again for a carrier before the packet is transmitted (see Figure 4.4 on page 49). If a node detects a carrier after deferring, the node will not begin transmitting, but will wait for no carrier and begin the deferring process again. This random-defer-then-check method helps to avoid the guaranteed collision of two nodes waiting for a third node to finish a packet transmission. The total number of times the Data Link Layer will defer for a particular packet transmission attempt is determined by the literal `max_defer_count` declared in the Data Link Layer software module (see Appendix H). If this value is exceeded, a fatal error occurs and the session is ungracefully terminated.

The Physical Layer is capable of receiving its own transmissions on the shared bus. The Data Link Layer uses this ability to detect collisions with another node's transmission. As the Data Link Layer transmits each character of a packet, it compares the received character to the character just transmitted. Normally the two are equal. When two nodes transmit simultaneously, though, the two electrical signals are superimposed and distorted. Neither of the nodes will receive back the character they
attempted to transmit. Thus, each node will detect the collision.

When a collision is detected, each node will abort the transmission of the remainder of its packet. Each node then backs off and attempts to retransmit its packet at a later time. Notice that the random defer before packet transmitting helps to avoid the collision of the two nodes again after the back off. The period of time the Data Link Layer will wait after a collision is determined by the literal `backoff_timer_restart` declared in the Data Link Layer software module (see Appendix H). The total number of times the Data Link Layer will attempt to retransmit a particular packet is determined by the literal `max_backoff_count`. If this value is exceeded, a fatal error occurs and the session is ungracefully terminated.

Notice that the collision detection mechanism can also detect transmission errors due to noise. Noise on the cable bus can also cause a mismatch between transmitted and received characters. In this case, the Data Link Layer will assume a collision occurred and will retransmit the packet, recovering from the error. However, not all transmission errors can be detected in this way. Noise can cause the remote node to incorrectly receive a character, while the local transmitting node does not detect the error. Thus the collision detection/recovery mechanism provides only limited transmission error recovery.

### 4.2.4 Implementation

The Data Link Layer is implemented in software as two separate interrupt-driven state machines: one for packet transmission with acknowledgement reception and one for
Table 4.2: Data Link State Variables

<table>
<thead>
<tr>
<th>State Name</th>
<th>State Variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>RW</td>
<td>packet_recv_wait</td>
</tr>
<tr>
<td>RC_11</td>
<td>packet_recv_char, char_count</td>
</tr>
<tr>
<td>RA_11</td>
<td>packet_recv_ack, char_count</td>
</tr>
<tr>
<td>TW</td>
<td>packet_trans_wait</td>
</tr>
<tr>
<td>TP</td>
<td>packet_trans_pending</td>
</tr>
<tr>
<td>TD</td>
<td>packet_trans_defer</td>
</tr>
<tr>
<td>TB</td>
<td>packet_trans_backoff</td>
</tr>
<tr>
<td>TC_11</td>
<td>packet_trans_char, char_count</td>
</tr>
<tr>
<td>TA</td>
<td>packet_trans_ack</td>
</tr>
</tbody>
</table>

packet reception with acknowledgement transmission (see Appendix H). Each state of the machines performs a step in the packet transmission/reception process. When an interrupt event occurs (e.g., the Physical Layer received a character or a carrier is detected), the particular action and next state taken is determined by the current state of the machines. The machines’ states are represented as variables within the software. Figure 4.6 depicts an overview of the packet receiver state machine and Figure 4.7 depicts an overview of the packet transmitter state machine. The unlabeled state transition arrows in the figures indicates the normal sequencing of states. The labeled state transitions indicates abnormal events in the process (e.g., carrier lost while receiving a packet, collision detected while transmitting). Table 4.2 lists the software variables that correspond to each state shown in the figures.
Figure 4.6: Overview of the Packet Receiver State Machine
Figure 4.7: Overview of the Packet Transmitter State Machine
4.3 Network Layer

Once a session is established, the Network Layers of two nodes provide an error-free block-oriented virtual circuit service between their corresponding Transport Layers. Using the packet/acknowledgement transaction services provided by the Data Link Layer, the Network Layer performs the necessary packet routing and maintenance to form a virtual circuit. The Transport Layer is not concerned with the necessary details. The Transport Layer can request the Network Layer to establish a circuit with a specified remote node, to terminate a current circuit, to transmit a block of characters over the circuit, or to receive a block of characters.

4.3.1 Packet and Acknowledgement Formats

Figure 4.8 shows the packet format used by the Network Layer. The first two characters of the packet represent the high and low order bytes of a 16-bit address, respectively. This address specifies the destination of the packet (the particular node which is to act upon the packet when received). The next pair of characters, similarly, specifies the source of the packet (the particular node which transmitted the packet). Next follows a single character control field which specifies the type of the packet. A packet is either a data packet (which contains user data) or a control packet (which controls aspects of the virtual circuit). Figure 4.9 shows the possible formats of the control field. Following the control field character of a packet is a single character which specifies the number of data characters present in the data block of the packet. For
control packets this data length character always has a zero value and it is the last character of the packet (no data block). For data packets this data length character specifies the number of characters in the data block which comprises the remainder of the packet. Although the data length character could specify lengths up to 255, the data length character value (and the number of characters in the data block) is limited to a maximum value of 122 characters to better utilize RAM buffer space. The actual size and contents of the data block in a particular data packet are determined by the Transport Layer.

When the Network Layer receives a packet, it can acknowledge the packet with one of several different acknowledgement characters (see Figure 4.10). Whether an acknowledgement is sent (and if sent what type of acknowledgement is sent), depends on the received packet's address and control fields, and on the current state of the node.

4.3.2 Routing

The routing of packets between nodes is achieved by having the Network Layer act upon only those packets received with a destination address which matches its own
<table>
<thead>
<tr>
<th>Character Format</th>
<th>Literal Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0, 0, 0, 0, 0, s, s</td>
<td>control.data</td>
</tr>
<tr>
<td>0, 0, 0, 0, 1, d, d</td>
<td>control_open_session</td>
</tr>
<tr>
<td>0, 0, 0, 1, 0, d, d</td>
<td>control_close_session</td>
</tr>
<tr>
<td>0, 0, 0, 1, 1, d, d</td>
<td>control_flow_off</td>
</tr>
<tr>
<td>0, 0, 1, 0, 0, d, d</td>
<td>control_watch_dog</td>
</tr>
</tbody>
</table>

ss = 2-bit sequence number
dd = do not care

Figure 4.9: Control Field Formats

<table>
<thead>
<tr>
<th>Character Format</th>
<th>Literal Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>0, 0, 0, 0, f, s, s</td>
<td>ack_ok (data)</td>
</tr>
<tr>
<td>0, 0, 0, 0, 0, 0, 0</td>
<td>ack_ok (control)</td>
</tr>
<tr>
<td>0, 0, 0, 0, 1, 0, 0</td>
<td>nack_not_available</td>
</tr>
<tr>
<td>0, 0, 0, 1, 0, 0, 0</td>
<td>nack_not_available</td>
</tr>
</tbody>
</table>

ss = 2-bit sequence number
f = flow control bit

Figure 4.10: Acknowledgement Character Formats

59
node's address. All other packets are ignored by the Network Layer. Thus, to send a packet to a particular node, the Network Layer specifies the remote node's address in the destination address field of the packet. A virtual circuit is achieved by having two Network Layers agree to address and send packets to each other. This agreement is established when one node sends a session establishment control packet to another node and the other node acknowledges positively. The source address field of a packet allows the receiving node to identify the source of the packet. Thus, the Network Layer is able to distinguish between packets sent by the remote node of the circuit or sent by some other node.

4.3.3 Packet/Acknowledgement Transaction

There are five different types of packets: one for data transmission and four for control purposes. The control field of the packet specifies the packet's type. Two types of control packets are used for session establishment and graceful session termination. When a node wishes to establish a session with a remote node, the Network Layer sends a control_open_session control packet (the control field has a value of control_open_session) to the remote node. The remote node will then acknowledge the control packet with one of three different acknowledgement characters depending on the remote node's current state. If the remote node is ready and available to establish a session, the remote node will respond with an ack_ok acknowledgement. At this point both nodes are in agreement and the session is established. If the remote node is busy with another session, the remote node will respond with nack_busy acknowledgement and no
session is established. If the remote node is not able to establish a session (e.g., DTR control is on), the remote node will respond with a *nack_not_available* acknowledgement. If a node wishes to gracefully terminate a session, the Network Layer sends a *control_close_session* packet to the remote node. The remote node then responds with an *ack_ok* acknowledgement and the session is gracefully terminated.

Once a session is established the two nodes can exchange data packets. The control fields of data packets have the value *control_data*. The 2-bit sequence number which is part of this value is used in the error recovery process. The data packets have blocks of data that the Transport Layer has requested to be sent over the virtual circuit. When a remote node receives a data packet, the remote node will respond with a special *ack_ok* acknowledgement character. This character contains the same 2-bit sequence number the received data packet has and a flow control bit. When the remote node cannot accept another data packet, the remote node sends an acknowledgement character with the flow control bit set. This invokes flow control, and the local node will stop sending data packets to the remote. The remote node may need to invoke flow control, for example, if its attached user device is operating at a slow baud rate and data packets are coming too fast. When the remote node is ready to receive more data packets, the remote node sends a *control_flow_off* control packet to the local node. This packet requests that the flow control be removed.

The final type of packet is the *control_watch_dog* control packet. This packet is sent periodically by the Network Layer to insure that the remote node is still active in the session. The remote node simply acknowledges this packet with an *ack_ok* acknowl-
edgement. Notice that if the remote node has ungracefully terminated the session, the remote would not acknowledge the packet. Thus, the local node will ungracefully terminate the session also.

4.3.4 Error Recovery

To impose an error-free virtual circuit on the non-error-free packet/acknowledgement services of the Data Link Layer, the Network Layer uses several techniques to recover from transmission errors. Since the Data Link Layer detects and discards any erroneous packets received, the Network Layer can assume that any packets it receives from the Data Link Layer are error-free. The Network Layer uses the acknowledgement transaction to guarantee a remote node receives the packet correctly. If the Network Layer does not receive a valid acknowledgement to a packet transmitted, the Network Layer assumes the remote node did not receive the packet correctly. To recover, the Network Layer retransmits the packet and waits for an acknowledgement again. The total number of times the Network Layer will retransmit a particular packet is determined by the literal `max_retry_count` declared in the Network Layer software module (see Appendix G). If this value is exceeded, a fatal error occurs. If a session is active at this time, it is ungracefully terminated.

This retransmit technique will allow recovery from erroneous packets, but problems can still occur. Consider the following scenario: The local node transmits a packet which the remote node receives correctly and acknowledges. Now, suppose the remote's acknowledgement is damaged by noise and the local Network Layer does not receive
it. The local Network Layer will assume the remote did not correctly receive the transmitted packet and will retransmit the packet. Suppose this second transaction attempt is successfully completed: the remote node receives the packet correctly again and the local node receives the remote's acknowledgement correctly. It appears that error recovery was successful. However, from the remote node's point of view the above scenario appears as two separate normal packet transactions. The remote node has received two copies of the same packet and it must recognize this in order not to send duplicate data to its attached user device.

The sequencing numbers in the control field of a data packet and the corresponding acknowledgement are used to avoid this duplication problem. As data packets are transmitted their control fields are consecutively numbered. If a packet is retransmitted the same sequence number is used in the control field. Thus, the remote node can detect duplicate packets by monitoring the sequence number of received packets. When the remote node receives a duplicate, the node acknowledges the packet normally but discards the duplicate packet. The only possible ambiguity is between a data packet and the preceding or succeeding data packets, not between the preceding and succeeding packets themselves. Therefore, a 1-bit sequence number is the minimum size necessary to detect duplication. Since the extra computational overhead is minimal, a 2-bit sequence number is used to provide redundancy for error detection.
4.3.5 Implementation

The Network Layer is implemented in software as three enhanced queue structures: one queue for control packets to be transmitted, one for data packets to be transmitted, and one for data packets received (see Appendix G). The queues provide a buffering between the packet transmission/reception process (in the Data Link Layer) and the packet generation/evaluation process (in the Transport and Network Layers). The two data queues link the Transport and Data Link Layers together. Data blocks enqueued by the Transport Layer are encapsulated. The resulting packet is then placed in a data packet queue for the Data Link Layer to eventually transmit. Data packets received by the Data Link Layer are enqueued on the other data queue. The Transport Layer can then dequeue the data blocks from these received packets. The control packets, which are generated by the Network Layer itself, are held in the control queue. These control packets are transmitted along with the data packets by the Data Link Layer. Acknowledgement characters are generated by the Network Layer when the Data Link Layer enqueues a received packet. Acknowledgement characters are evaluated by the Network Layer when the Data Link Layer requests the next packet to transmit.

4.4 User, Command, and Transport Layers

The User Layer handles the interface between the user device and the node. The User Layer provides character transmission and reception services to the Transport and Command Layers. The character format, timing and electrical characteristics are
implemented in hardware (see Chapter 3, Section 3.1.4). Flow control and Autobaud Mode are implemented in software (see Appendix C). The specific features of the User Layer have been discussed in Chapter 2.

The Command Layer handles the command dialog between the node and the attached user device when the node is in Command Mode. Using the services provided by the User Layer, the Command Layer interprets and executes commands from the user device. The Command Layer is implemented in two software modules (Appendices E and F). The first module decodes a command and parameters entered by using a look-up table. Once the command has been uniquely identified and confirmed, an appropriate procedure in the second software module is called to execute the command. The specific features of the Command Layer have been discussed in Chapter 2.

The Transport Layer handles the blocking and unblocking of user data and the switching of the node between Connect and Command Modes. Using the timer buffering technique (see Chapter 2, Section 2.5.2) the Transport Layer gathers individual characters from the User Layer into blocks. These data blocks are then given to the Network Layer for transmission to the remote node. The Transport Layer also takes data blocks from the Network Layer and sends the data character by character to the User Layer. When the node is reset, the Transport Layer invokes the hardware self tests and initializes the operating attributes of the node. The specific features of the Transport Layer have been discussed in Chapter 2.
Chapter 5

Conclusions

The ADSL Local Area Network was conceived with four design goals in mind: low cost, versatile user interface, fault tolerance, and maintainability. Certainly, the low cost has been achieved. A single twisted pair cable is all that is needed for the system’s foundation. To connect a device to the system requires a node unit which consists of a small number of common “off-the-shelf” components. All node units are functionally identical: the same hardware and software. Therefore, mass production techniques can be applied to further reduce the cost. The low cost of the node unit was achieved primarily by pushing most of the node’s functionality into software, leaving general purpose hardware behind. As a result, the controlling software of the node required large quantities of time and effort to develop. However, since the software is now developed it can be duplicated essentially without cost.

A disappointing result of implementing most of the functionality in software is reduced bit transmission rate on the cable bus. Although the hardware circuitry can operate four times faster, the complexity of the software has forced the use of a reduced bit rate on the cable bus. The CSMA/CD and encapsulation techniques require
bandwidth for control purposes, thus further reducing the actual bandwidth available for data communication. Though still faster than the maximum bit rate coming from a user device, this reduced bandwidth means that fewer simultaneous virtual circuits can be supported before congestion occurs on the cable bus. If flow control is properly used when congestion occurs, then reliable communication is still possible; however, it is slower.

The user interface on the node unit has several operating attributes that can be individually set to match that of the attached user device. This versatility, however, can make the node unit difficult to use. As more operating attributes are supported, allowing a wider variety of communication interfaces, there are more parameters that must be determined and set before communication is possible. The Autobaud Mode is designed to help alleviate this problem by making the setting of operating attributes more automatic. As a result of the layered architecture, a node unit with a different type of user interface (e.g., parallel) can be easily constructed. Changes necessary for a new interface would be mainly confined to the User Layer. All other layers would require little or no change.

By distributing the control and keeping the shared components passive, a high degree of reliability has been achieved in the ADSL LAN system. Failure of a node unit will tend to affect the communication ability of only that particular node. From the beginning, the virtual circuit service was designed to provide error-free communication, tolerant of noise. A wide variety of techniques are used to achieve this (e.g., differential transceivers for noise rejection, packet retransmission for error recovery, and data

67
flow control). Communication errors either are recoverable or are avoided entirely. Although the LAN cannot avoid or recover from every possible error, it appears that the LAN provides circuits that are as error-free as those in point-to-point networks.

The distributed design and low node unit cost make maintenance of the ADSL LAN easy. When a node unit has failed, it can simply be replaced with another node unit, since node units are functionally identical. The faulty node can then be repaired as time permits without loss of communication on the LAN. As with any large system, the management of the LAN is an important aspect. For example, addresses of the nodes in any given ADSL LAN system should be carefully and consistently assigned to the nodes. No two nodes should have the same address, and the LAN would be of no use if the user does not know the addresses of the nodes. To avoid these problems, a careful record of the LAN's configuration should be kept.

Although not a high performance network, the ADSL Local Area Network provides a comprehensive and flexible method of interconnecting computer systems and peripheral equipment at low cost. Flexibility and physical simplicity are the ADSL LAN's primary advantages over the point-to-point network. Only one cable and a few nodes are needed for a tremendous ability to communicate.
Appendix A

Hardware Schematics and Components

Table A.1: Component Table

<table>
<thead>
<tr>
<th>Reference Designation</th>
<th>Component Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IC1</td>
<td>8032 Microprocessor</td>
</tr>
<tr>
<td>IC2</td>
<td>74LS373 LS TTL Octal D-Type Latch</td>
</tr>
<tr>
<td>IC3</td>
<td>27128 EPROM, 200ns</td>
</tr>
<tr>
<td>IC4</td>
<td>HM6116 CMOS RAM, 200ns</td>
</tr>
<tr>
<td>IC5, IC6, IC7</td>
<td>74LS156 LS TTL Open-Collector Demultiplexer</td>
</tr>
<tr>
<td>IC8</td>
<td>INS8250A Async Communications Element</td>
</tr>
<tr>
<td>IC9</td>
<td>7404 TTL Hex Inverter</td>
</tr>
<tr>
<td>IC10</td>
<td>75188 or MC1488 Quadruple RS-232-C Line Driver</td>
</tr>
<tr>
<td>IC11</td>
<td>75189 or MC1489 Quadruple RS-232-C Line Receiver</td>
</tr>
<tr>
<td>IC12</td>
<td>75176 Differential Bus Transceiver</td>
</tr>
<tr>
<td>IC13</td>
<td>75174 Quadruple Differential Line Driver</td>
</tr>
<tr>
<td>IC14</td>
<td>75175 Quadruple Differential Line Receiver</td>
</tr>
<tr>
<td>CY1</td>
<td>12.000 MHz Crystal</td>
</tr>
<tr>
<td>CY2</td>
<td>1.8432 MHz Crystal</td>
</tr>
<tr>
<td>LED1</td>
<td>T1 3/4 LED, Red</td>
</tr>
</tbody>
</table>

Note: Either a 75176 is used or a 75174 and a 75175 are used, but not all three.
Figure A.1: Edge Connector Schematic
Figure A.2: Processor Block Schematic

71
Figure A.3: Memory Block Schematic
Figure A.4: Switch Bank Block Schematic
Figure A.5: User Interface Block Schematic
Figure A.6: Transceiver Block Schematic
Appendix B
Hardware Diagnostic Software

PL/M-51 COMPILER

HTEST V2.0 LAN Hardware Confidence Test

IBIS-II PL/M-51 V1.1
COMPILER INVOKED BY: PLM51 :F7:HTEST2.JMK

$PAGEWIDTH( 95 )
$title(' HTEST V2.0 LAN Hardware Confidence Test ')
$ROM( LARGE )
$KEY( on_chip_8751 )
$KEY( on_chip_ice81 )
$RESET( off_chip_2764 )
$RESET( off_chip_27128 )

1 1 htest: DS;
/---------------------------------------------------------------------------------------------------------/
/                                                                                                          /
/ 1                                                                                                      /
/                                                                                                          /
/ Hardware diagnostic program. Allows hardware confidence testing of                                          /
/ a LAN node controller card.                                                                             /
/                                                                                                          /
/ Dave WicKliff                                                                                           /
/ Last Modified: 11/27/85                                                                                  /
/                                                                                                          /
/ $IF on_chip_8761                                                                                         /

2 1 DECLARE version LITERALLY "'2.0 On Chip 8761'" ;
3 1 DECLARE htest_check_sum WORD AT( OFFEH ) CONSTANT( OFFFFH ); /= 4x =/
$ENDIF

76
PL/M-61 Compiler  BTEST V3.0 LAW Hardware Confidence Test

$EJECT
$IF on_chip_ics61
DECLARE version LITERALLY "'2.0 On Chip IC61"';
DECLARE htest_check_sum WORD AT( 1FFFH ) CONSTANT( OFFFFH ); /* 8k */
$ENDIF

$IF off_chip_2764
DECLARE version LITERALLY "'2.0 Off Chip 2764"';
DECLARE htest_check_sum WORD AT( 1FFFH ) CONSTANT( OFFFFH ); /* 8k */
$ENDIF

$IF off_chip_27128
DECLARE version LITERALLY "'2.0 Off Chip 27128"';
DECLARE htest_check_sum WORD AT( 3FFFH ) CONSTANT( OFFFFH ); /* 16k */
$ENDIF

$INCLUDE ( :f7:io8260.dcl )
= /****************************
= / Declarations for the procedures and global variables for module 108260.PSI
=

4 1 = DECLARE swi_image BYTE EXTERNAL;
6 1 = DECLARE sw2_image BYTE EXTERNAL;
8 1 = DECLARE user_initialize: PROCEDURE( data ) EXTERNAL;
9 2 = DECLARE data BYTE; END;
11 2 = DECLARE user_switches: PROCEDURE EXTERNAL; END;
13 2 = DECLARE user_set_band: PROCEDURE( data ) EXTERNAL;
14 2 = DECLARE data BYTE; END;
16 2 = DECLARE user_in_byte_status: PROCEDURE BIT EXTERNAL; END;
18 2 = DECLARE user_in_byte: PROCEDURE BYTE EXTERNAL; END;
20 2 = DECLARE data BYTE; END;
21 2 = DECLARE data BYTE; END;
23 1 = DECLARE cr LITERALLY '0Dh'; /* Carriage Return */
24 1 = DECLARE crf LITERALLY '0Dh, 0AH'; /* Carriage Return, Line Feed */
26 1 = DECLARE well LITERALLY '08h'; /* Bell */
28 1 = DECLARE ram_small LITERALLY '0800h'; /* Size of small External RAM */
29 1 = DECLARE ram_large LITERALLY '2000h'; /* Size of large External RAM */
30 1 = DECLARE prom_small LITERALLY '2000h'; /* Size of small External EPROM */
31 1 = DECLARE prom_large LITERALLY '4000h'; /* Size of large External EPROM */
32 1 = DECLARE command BYTE;
33 1 = DECLARE test_number BYTE;
34 1 = DECLARE ram_size BIT;
35 1 = DECLARE prom_size BIT;
36 1 = DECLARE quiet BIT;
37 1 = DECLARE forever BIT;
38 1 = DECLARE fail BIT;
39 1 = DECLARE loop_data_fail BIT;
40 1 = DECLARE loop_crs, dtr, fail BIT;
41 1 = DECLARE looptabl_fail BIT;
DECLARE status lookout( 16 ) BYTE;

message: PROCEDURE(str_p);

DECLARE str_p ADDRESS;
DECLARE char BASED str_p BYTE CONSTANT;

DO WHILE char <> 0;
    CALL user_out_byte( char );
    str_p = str_p + 1;
END;

display_hex: PROCEDURE( data );

DECLARE (data, temp) BYTE;

temp = SHR(data,4);
IF temp > 9 THEN CALL user_out_byte(temp + 'A' - 10);
ELSE CALL user_out_byte(temp + '0');

temp = data AND DFH;
IF temp > 9 THEN CALL user_out_byte(temp + 'A' - 10);
ELSE CALL user_out_byte(temp + '0');

DO CASE test_number;
    CALL message((', '0 - Flash Status LED Test ', 0 ));
    CALL message((', '1 - External Interrupt 1 Test ', 0 ));
    CALL message((', '2 - INSS260 Internal Loopback Test ', 0 ));
    CALL message((', '3 - INSS260 External Loopback Test ', 0 ));
    CALL message((', '4 - INSS260 External Loopback Test ', 0 ));
    CALL message((', '5 - Read Switch Banks Test ', 0 ));
    CALL message((', '6 - External RAM Read/Write Test ', 0 ));
    CALL message((', '7 - External RAM Uniqueness Test ', 0 ));
    CALL message((', '8 - External RAM Valid Test ', 0 ));
PL/N-81 COMPILER

HTEST V2.0 LAW Hardware Confidence Test

73 3 CALL message(. '0 - External EPROM Checksum Test', 0));
74 3 CALL message(. 0));
75 3 CALL message(. 0));
76 3 CALL message(. 0));
77 3 CALL message(. 0));
78 3 CALL message(. 0));
79 3 CALL message(. 0));
80 3 END;
81 1 END display_result;
82 2 display_result: PROCEDURE;
83 2 IF fail THEN CALL message(. 'Failed!', crlf, 0));
84 2 ELSE CALL message(. 'Passed', crlf, 0));
85 1 END display_result;
86 1 help: PROCEDURE;
87 2 IF fail THEN CALL message(. 'Failed!', crlf, 0));
88 2 ELSE CALL message(. 'Passed', crlf, 0));
89 2 command = user_in_byte;
90 2 CALL message(. crlf, crlf, 'Available commands:',
91 2 crlf, 'Dm - Disable test a',
92 2 crlf, 'Em - Enable test a',
93 2 crlf, 'F - Run all enabled tests forever',
94 2 crlf, 'R - This help menu',
95 2 crlf, 'P - Toggle External EPROM size',
96 2 crlf, 'Q - Toggle quiet mode on/off',
97 2 crlf, 'R - Toggle External RAM size',
98 2 crlf, 'S - Show status of available test',
99 2 crlf, 'T - Run all enabled tests',
100 2 Tn - Run test a'; crlf, 0));
101 1 END help;
102 2 status: PROCEDURE;
103 2 IF quiet THEN CALL message(. 'ON', crlf, 0));
104 2 ELSE CALL message(. 'OFF', crlf, 0));
105 1 display: PROCEDURE;
106 2 IF quiet THEN CALL message(. 'OFF', crlf, 0));
107 2 ELSE CALL message(. 'ON', crlf, 0));
CALL message(. ( crlf, ' External RAM size: ', 0 ));
CALL message(. ( '8K bytes', crlf, 0 ));
ELSE CALL message(. ( '2K bytes', crlf, 0 ));
CALL message(. ( crlf, ' External EPROM size: ', 0 ));
IF eeprom_size
THEN CALL message(. ( '16K bytes', crlf, crlf, 0 ));
ELSE CALL message(. ( '8K bytes', crlf, crlf, 0 ));
DO test_number=0 TO 15;
IF status_lookup( test_number ) <> 0
THEN DO;
CALL display_name;
IF status_lookup( test_number ) = 1
THEN CALL message(. ( 'ENABLED', crlf, 0 ));
ELSE IF status_lookup( test_number ) = 2
THEN CALL message(. ( 'DISABLED', crlf, 0 ));
ELSE CALL message(. ( 'NOT AVAILABLE', crlf, 0 ));
END;
END;
CALL message(. ( crlf, crlf, 0 ));

END status;

test_power_up: PROCEDURE;
/*--------------------------*/
/* test_power_up */
/* */
/* Test procedure for power-up or reset. */
usr_status_led = 0;
/* Set User Port to 8 bit word, disable parity, and 9600 baud */
swi_image = 00000000;
CALL usr_initialize( swi_image );
fail = 0;
END test_power_up;
flash_led: PROCEDURE;
/*--------------------------*/
/* flash_led */
/* Procedure to flash the User Status LED. */
DECLARE i BYTE;
DO i=1 to 10;
user_status_led = 1; /* Turn LED on */
CALL TIME( 255 ); /* for about 0.6 sec */
CALL TIME( 255 );
PL/M-81 COMPILDE  NTST V2.0 LAM Hardware Confidence Test

131 3  user_status_led = 0;  /* Turn LED off */
132 3  CALL TIME( 265 );  /* for about 0.5 sec */
133 3  CALL TIME( 255 );
134 3  END;

135 1  END flash_led;

136 2  test_flash_led: PROCEDURE;
/*-------------------------------*/
/* test_flash_led */
/* Test procedure to flash the User Status LED. */

137 2  IF NOT ( forever AND quiet )
    THEN DO;
        CALL display_name;
    END;
    CALL flash_led;

138 2  fail = 0;

139 1  END test_flash_led;

140 2  test_interrupt_i: PROCEDURE;
/*-------------------------------*/
/* test_interrupt_i */
/* Test procedure to check if the External Interrupt 1 input is always low. */

141 2  $IF NOT on_chip_6781
    DECLARE i BYTE;
    DECLARE p3_3 BIT AT( 083H ) REGISTER;
    IF NOT ( forever AND quiet ) THEN CALL display_name;

142 2  fail = 0;
    DO i=0 to 100;
        IF p3_3 = 1 THEN fail = 1;
    END;

143 2  IF fail AND quiet AND forever THEN CALL display_name;
    IF fail OR NOT( forever AND quiet ) THEN CALL display_result;

$ENDIF

144 1  END test_interrupt_i;

145 2  ins8250_loopback: PROCEDURE;
/*-------------------------------*/
/* ins8250_loopback */
/* Procedure to perform loopback tests on both the data path and the modem */
/* control/status path in (or through) the 1NS8250. */

146 2  DECLARE 1 BYTE;
147 2  DECLARE modem_control_reg BYTE AT( 8000H + 4 ) AUXILIARY;
148 2  DECLARE line_status_reg BYTE AT( 8000H + 5 ) AUXILIARY;

81
DECLARE modem_status_reg BYTE AT(4000H + 0) AUXILIARY;

fail = 0;
loop_data_fail = 0;
loop_cts_dtr_fail = 0;
loop_der_rts_fail = 0;
loop_rri_out_fail = 0;

/* Test data path */
DO i=0 to 255;
    CALL out_byte( i );
    DO WHILE ( line_status_reg AND 01000000B )=0: END;
    IF ( in_byte_status
        THEN DO;
            IF ( in_byte <> i )
                THEN DO:
                    fail = 1;
                    loop_data_fail = 1;
                    END;
            END;
    ELSE DO;
        END;
END;

/* Test modem control/status path */
    IF ( modem_control_reg OR 00000001B ) = 0
        THEN DO:
            fail = 1;
            loop_cts_dtr_fail = 1;
            END;
    modem_control_reg = ( modem_control_reg AND 1111110B )
    IF ( modem_status_reg AND 00100000B ) <> 0
        THEN DO:
            fail = 1;
            loop_cts_dtr_fail = 1;
            END;
    modem_control_reg = ( modem_control_reg OR 0000001B )
    IF ( modem_status_reg AND 00010000B ) = 0
        THEN DO:
            fail = 1;
            loop_der_rts_fail = 1;
            END;
    modem_control_reg = ( modem_control_reg AND 11111101B )
    IF ( modem_status_reg AND 00010000B ) <> 0
        THEN DO:
            fail = 1;
            loop_der_rts_fail = 1;
            END;
    modem_control_reg = ( modem_control_reg OR 00000001B )
    IF ( modem_status_reg AND 00010000B ) = 0
        THEN DO:
            fail = 1;
PL/M-51 COMPILER

HTEST V2.0 LAN Hardware Confidence Test

202 3 loop戚ati fail = 1;
203 3 END;
204 2 modem_control_reg = ( modem_control_reg AND 11111011B );
205 2 IF ( modem_status_reg AND 01000000B ) <> 0
206 3 THEN DO;
207 3 fail = 1;
208 3 loop戚ati fail = 1;
209 3 END;

210 1 END ins250_loopback;
211 2 test_8280_internal_loop: PROCEDURE;
212 2 DECLARE modem_control_reg BYTE AT ( 8000H + 4 ) AUXILIARY;
213 2 DECLARE line_status_reg BYTE AT ( 8000H + 5 ) AUXILIARY;
214 2 IF NOT ( forever AND quiet ) THEN CALL display_name;
215 2 DO WHILE ( line_status_reg AND 01000000B )=0; END;
216 2 modem_control_reg = modem_control_reg OR 00010000B;
217 2 CALL ins250_loopback;
218 2 modem_control_reg = modem_control_reg AND 11101111B;
219 2 IF fail AND forever AND quiet THEN CALL display_name;
220 2 IF fail OR NOT(forever AND quiet) THEN CALL display_result;
221 2 IF fail THEN DO:
222 3 IF loop_data_fai1
223 3 THEN CALL message( ( " Data path failed", crlf, 0 ));
224 3 IF loop_cts_dtr_fai1
225 3 THEN CALL message( ( " CTS/DTR path failed", crlf, 0 ));
226 3 IF loop_dsr_rts_fai1
227 3 THEN CALL message( ( " DSR/RTS path failed", crlf, 0 ));
228 3 IF loop戚ati_fai1
229 3 THEN CALL message( ( " RI/OUT path failed", crlf, 0 ));
230 3 END;
231 1 END test_8280_internal_loop;
232 2 test_8280_external_loop: PROCEDURE;
233 2 DECLARE line_status_reg BYTE AT ( 8000H + 5 ) AUXILIARY;
234 2 DECLARE word_reg;
235 2 DECLARE word_reg;
236 2 CALL display_name;
237 2 CALL message( ( " Remove terminal connection and connect an external loopback connector", crlf, " within 20 seconds. Test is over when User Status LED goes off.", crlf, " Then reconnect the terminal and type a <CR>.", crlf, 0 ));

83
PL/M-64 COMPILER

RTTEST V3.0 LAN Hardware Confidence Test

DO i = 1 TO 256;
CALL TIME( 260 );
END;

usr_status_led = 1;
DO WHILE ( line_status_reg AND 0100000000 ) = 0: END;
CALL line_SISO_loopback;
usr_status_led = 0;

DO WHILE user_in_bytes<>cr: END;

CALL display_name;
CALL display_result;
IF fail THEN DO;
IF loop_data_fail
THEN CALL message( " Data path failed", crlf, 0 );
IF loop_cts_dtr_fail
THEN CALL message( " RTS/DSR path failed", crlf, 0 );
IF loop_dsr_rts_fail
THEN CALL message( " DTR/CTS path failed", crlf, 0 );
IF loop_cdc_fail
THEN CALL message( " TEST/DCD path failed", crlf, 0 );
END;

$ENDIF

END test_(250)_external_loop:

238 1
test_display_switch: PROCEDURE;

/*--------------------------------------------*/
/* test_display_switch */
/* Procedure to read and display the three Switch Banks. */

$IF NOT on_chip_8751

display_bit: PROCEDURE( data );

/*--------------------------------------------*/
/* display_bit */
/* Procedure to display variable data as a bit sequence. */

DECLARE( i, data ) BYTE;

DO i = 1 TO 8;
CALL usr_out_byte( ' ' );
IF data < 128 THEN CALL usr_out_byte( '0' );
ELSE CALL usr_out_byte( '1' );
data = ROL( data, 1 );
END; /* DO*/

END display_bit;

IF NOT( forever AND quiet ) THEN DO:
CALL display_name;
CALL message( ( crlf, " bit 8 7 6 5 4 3 2 1", 0 ) );

84
CALL user_get_switches;

CALL message(. crlf, ' Bank 1 ', 0 );
CALL display_bit( sw1_image );
CALL message(. ' ', 0 );
CALL display_hex( sw1_image );

CALL message(. ' h ', crlf, ' Bank 2 ', 0 );
CALL display_bit( sw2_image );
CALL message(. ' ', 0 );
CALL display_hex( sw2_image );

CALL message(. ' h ', crlf, ' Bank 3 ', 0 );
CALL display_bit( sw3_image );
CALL message(. ' ', 0 );
CALL display_hex( sw3_image );
CALL message(. ' h ', crlf, 0 );

END;

$ENDIF

240 1 END test_display_switch;

241 2 test_ram_read_write: PROCEDURE;
/--------------------------------------------------------------------------/
/* test_ram_read_write */
/* Procedure to test the external RAM by simple read/write. */
/--------------------------------------------------------------------------/
242 2 DECLARE data BYTE;
243 2 DECLARE i WORD;
244 2 DECLARE j WORD;
245 2 IF NOT ( forever AND quiet ) THEN CALL display_name;

246 2 i = O;
247 2 fail = O;
248 2 IF ram.size
249 2 THEN j = ram_large - 1;
250 2 ELSE j = ram_small - 1;
251 2 DO WHILE i < j AND NOT fail;
252 3 data = not memory( i );
253 3 memory( i ) = data;
254 3 IF memory( i ) <> data
255 4 THEN do;
256 5 fail = 1;
257 5 IF ( forever AND quiet ) THEN CALL display_name;
258 5 CALL display_result;
259 5 CALL message(. ' at ', 0 );
260 5 CALL display_hex( HIGH( i ) );
261 5 CALL display_hex( LOW( i ) );
262 5 CALL message(. ' h ', crlf, 0 );
263 5 END;
264 4 ELSE i = i + 1;
265 4
END; /* DO WHILE */

IF NOT (fail OR (.forever AND quiet)) THEN CALL display_result;

END test_ram_read_write;

test_ram_uniquenes: PROCEDURE;

DECLARE data BYTE;
DECLARE i WORD;
DECLARE j WORD;

IF NOT (.forever AND quiet) THEN CALL display_name;

i = 0;
fail = 0;
IF ram_size
THEN j = ram_large - 1;
ELSE j = ram_small - 1;

DO i = 0 TO j;
memory( i ) = OAAB;
END;

DO WHILE i <= j AND NOT fail;

i = 0;
fail = 0;

DO WHILE i <= j AND NOT fail;

IF memory( i ) <> OAAB:
THEN DO;
fail = i;
IF (.forever AND quiet) THEN CALL display_name;
CALL display_result;
CALL message( (. ' at ', 0 ));
CALL display_hex( HIGH( i ));
CALL display_hex( LOW( i ));
CALL message( (. 'H', crlf, 0 ));
fail = i;
END;
ELSE DO;
memory( i ) = NOT OAAB;
i = i + 1;
END;
END; /* DO WHILE */

IF NOT (fail OR (.forever AND quiet)) THEN CALL display_result;

END test_ram_uniquenes;

test_eprom_checksum: PROCEDURE;

DECLARE data BYTE;
DECLARE i WORD;
DECLARE j WORD;

IF NOT (fail OR (.forever AND quiet)) THEN CALL display_result;

END test_eprom_checksum;

END test_eprom_checksum;

/ Procedure to test the external EPROM by computing a word checksum on */
DECLARE checksum WORD;
DECLARE i WORD;
DECLARE j WORD;
DECLAREeprom BASED i BYTE CONSTANT;
DECLARE offset LITERALLY 'eprom_large';

IF NOT ( forever AND quiet ) THEN CALL display_name;

checksum = 0;
i = offset;

IF sprom_size
  THEN j = sprom_large - 3 * offset;
  ELSE j = sprom_small - 3 * offset;

DO WHILE i <= j AND NOT fail;
  checksum = checksum + sprom;
  i = i + 1;
END;

checksum = checksum;

IF HIGH( checksum ) = sprom;
  THEN DO;
    i = i + 1;
    IF LOW( checksum ) = sprom
      THEN fail = 0;
      ELSE fail = 1;
  END;
ELSE fail = 1;

IF fail AND forever AND quiet THEN CALL display_name;
IF fail OR NOT ( forever AND quiet ) THEN CALL display_result;

IF fail THEN DO;
  CALL message (. 'computed - ' , 0 );
  CALL display_hex( HIGH( checksum ) );
  CALL display_hex( LOW ( checksum ) );
  CALL message (. 'H, read - ' , 0 );
  CALL display_hex( sprom );
  i = i + 1;
  CALL display_hex( sprom );
  CALL message (. 'H', crlf, 0 );
END;

END test_sprom_checksum;

disable_test: PROCEDURE;
/**************************************************************************/
/* disable_test */
/* Procedure to disable a specified test from running in "all test" or */
/* "forever test". */
**************************************************************************/
command = user_is_byte;
CALL user_out_byte( command );
IF command > 'Z' THEN command = command - 20H; /* Force to upper case */
IF command > '9'
    THEN test_number = command - 'A';
ELSE test_number = command - '0';

command = user_in_byte;
CALL message(( crlf, 0 ));

IF status_look_up( test_number ) <> 0
    THEN DO;
        CALL display_name;
        status_look_up( test_number ) = 2;
        CALL message(( 'DISABLED', crlf, 0 ));
    END;
END disable_test;

enable_test: PROCEDURE;
/**---------------------------------------------*/
/* enable_test */
/* Procedure to enable a specified test for running in "all test" or */
/* "forever test" */
/**---------------------------------------------*/

command = user_in_byte;
CALL user_out_byte( command );
IF command > '2'
    THEN command = command - 20H; /* Force to upper case */
ELSE test_number = command - 'A';
command = user_in_byte;
CALL message(( crlf, 0 ));

IF status_look_up( test_number ) <> 0
    THEN DO;
        CALL display_name;
        status_look_up( test_number ) = 1;
        CALL message(( 'ENABLED', crlf, 0 ));
    END;
END enable_test;

run_look_up: PROCEDURE;
/**---------------------------------------------*/
/* run_look_up */
/* Procedure to run the test specified by test_number. */
/**---------------------------------------------*/

CALL test_flash_led; /* Test 0 */
CALL testinterrupt_1;
CALL test_8250_internal_loop;
CALL test_8250_external_loop; /* Test 3 */
CALL null; /* Test 4 */
CALL test_display_switch; /* Test 5 */
CALL test_ram_read_write; /* Test 6 */
CALL test_ram_uniqueness; /* Test 7 */
CALL null; /* Test 8 */
CALL test��m_checksum; /* Test 9 */
CALL null; /* Test 10 */
CALL null; /* Test 11 */
CALL null; /* Test 12 */
CALL null; /* Test 13 */
CALL null; /* Test 14 */
CALL null; /* Test 15 */
END;

END run_look_up;

forever_test: PROCEDURE;
episode -------------------------------------------------------------;
/* forever_test */
/* Procedure to run all enabled tests continuously. */
command = user_in_byte;
IF command = cr
THEN DO;
    CALL message(.( crlf, 0 ));
    /* Run all enabled tests forever */
    forever = 1;
DO WHILE 1;
    DO number = 0 TO 15;
    IF status_look_up( test_number ) = 1 THEN CALL run_look_up;
END;
END:

END forever_test;

run_test: PROCEDURE;
episode -------------------------------------------------------------;
/* run_test */
/* Procedure to run either a specified test or all enabled tests once. */
command = user_in_byte;
CALL user_out_byte( command );
IF command > 'Z' THEN command = command - 20H; /* Force to upper case */
IF command = cr
THEN DO;
    CALL message(.( crlf, 0 ));
    /* Run all enabled tests */
    DO number = 0 TO 15;
        IF status_look_up( test_number ) = 1 THEN CALL run_look_up;
    END;
END;

ELSE DO;
/* Else run the individual test */
ELSE DO;
    IF command > '9'
        THEN test_number = command - 'A';
    ELSE test_number = command - '0';
    IF user_in_byte = cr
        THEN DO;

END;

89
CALL message(. ( crlf, 0 ));
END run_look_up;
END;

END run_test;
toggle_srom: PROCEDURE;
/*
= toggle_srom
/* Procedure to toggle between two sizes of External EPROM.
*/
call = user_in_byte;
srom_size = NOT srom_size;
call message(. ( crlf, ' External EPROM size: ', 0 ));
IF srom_size
THEN call message(. ( '16K bytes', crlf, 0 ));
ELSE call message(. ( '8K bytes', crlf, 0 ));
END toggle_srom;
toggle_quiet: PROCEDURE;
/*
toggle_quiet
/* Procedure to toggle between quiet mode and non-quiet mode.
*/
call = user_in_byte;
quiet = NOT quiet;
call message(. ( crlf, ' Quiet Mode ', 0 ));
IF quiet
THEN call message(. ( 'ON', crlf, 0 ));
ELSE call message(. ( 'OFF', crlf, 0 ));
END toggle_quiet;
toggle_ram: PROCEDURE;
/*
toggle_ram
/* Procedure to toggle between two sizes of External RAM.
*/
call = user_in_byte;
ram_size = NOT ram_size;
call message(. ( crlf, ' External RAM size: ', 0 ));
IF ram_size
THEN call message(. ( '8K bytes', crlf, 0 ));
ELSE call message(. ( '2K bytes', crlf, 0 ));
END toggle_ram;

/* Main Code */
/* Initialize test enable status */
$IF on_chip_8761
status_look_up( 0 ) = 1;
status_look_up( 1 ) = 3;
status_look_up( 2 ) = 1;
status_look_up( 3 ) = 3;
status_look_up( 4 ) = 0;
status_look_up( 5 ) = 3;
status_look_up( 6 ) = 1;
status_look_up( 7 ) = 1;
status_look_up( 8 ) = 0;
status_look_up( 9 ) = 1;
status_look_up( 10 ) = 0;
status_look_up( 11 ) = 0;
status_look_up( 12 ) = 0;
status_look_up( 13 ) = 0;
status_look_up( 14 ) = 0;
status_look_up( 15 ) = 0;

$ELSEIF
status_look_up( 0 ) = 1;
status_look_up( 1 ) = 1;
status_look_up( 2 ) = 1;
status_look_up( 3 ) = 2;
status_look_up( 4 ) = 0;
status_look_up( 5 ) = 2;
status_look_up( 6 ) = 1;
status_look_up( 7 ) = 1;
status_look_up( 8 ) = 0;
status_look_up( 9 ) = 1;
status_look_up( 10 ) = 0;
status_look_up( 11 ) = 0;
status_look_up( 12 ) = 0;
status_look_up( 13 ) = 0;
status_look_up( 14 ) = 0;
status_look_up( 15 ) = 0;

$ENDIF

ram_size = 0;
sprom_size = 0;
quiet = 0;
forever = 0;

CALL test_power_up;

IF fail
   THEN DO WHILE 1; CALL flash_led; END;

CALL flash_led;

CALL message(( crlf, 'LAW Node Hardware Confidence Test - Version ', version, crlf, ( Enter H for help )', crlf, crlf, 0 ));

DO WHILE 1;
CALL message(( celf, '>', 0 ));
command = user_in_byte;
CALL user_out_byte( command );
IF command = '2' THEN command = command - 20H; /* Force to upper case */

IF command = 'D' THEN CALL disable_test;
ELSE IF command = 'E' THEN CALL enable_test;
ELSE IF command = 'F' THEN CALL forever_test;
ELSE IF command = 'H' THEN CALL help;
ELSE IF command = 'P' THEN CALL toggleeprom;
ELSE IF command = 'Q' THEN CALL togglequiet;
ELSE IF command = 'R' THEN CALL toggleram;
ELSE IF command = 'S' THEN CALL status;
ELSE IF command = 'T' THEN CALL run_test;
ELSE CALL help;

END; /* DO WHILE */
END htest;

MODULE INFORMATION:
(STATIC-OVERLAYABLE)
CODE SIZE = 0067H 2391D
CONSTANT SIZE = 0806H 1286D
DIRECT VARIABLE SIZE = 12H+08H 18D+ 8D
INDIRECT VARIABLE SIZE = 00H+00H 0D+ 0D
BIT SIZE = 08H+00H 9D+ 0D
BIT-ADDRESSABLE SIZE = 00H+00H 0D+ 0D
AUXILIARY VARIABLE SIZE = 0000H 0D
MAXIMUM STACK SIZE = 0000H 16D
REGISTER-BANK(S) USED:
0
899 LINES READ
0 PROGRAM ERROR(S)
END OF PL/N-61 COMPILATION
Appendix C

User Layer Module

PL/N-51 COMPILER  USRLJ - User Interface Layer Module

ISIS-II PL/N-51 V1.1
COMPILER INVOKED BY: PLM51 :PR:USRLJ.JNK

$PAGEWIDTH(96)
$TITLE('USRLJ - User Interface Layer Module ')
$REGISTERBANK(2)

1 1 overlay: DO;
/--------------------------------------------------------------------------/
1 1 /* usrv */
1 1 /**/
1 1 /* Dave Wickliff */
1 1 /* Last Modified: 11/27/86 */
1 1 /**/
1 1 $NOLIST

/--------------------------------------------------------------------------/
1 1 /* Externals from TIMERV */
1 1 DECLARE buffer_timer BYTE EXTERNAL;

/--------------------------------------------------------------------------/
1 1 /* Externals from TLAY */
1 2 check_error: PROCEDURE EXTERNAL USING 0; END;
1 8 1 DECLARE session_escape_sequence WORD EXTERNAL;
1 9 1 DECLARE session_escape_state BIT EXTERNAL;
$JECT
//+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
/* Local Definitions */

/* INSG580 registers */
  DECLARE data_reg BYTE AT (8000H + 0) AUXILIARY;
  DECLARE interrupt_enable_reg BYTE AT (8000H + 1) AUXILIARY;
  DECLARE interrupt_id_reg BYTE AT (8000H + 2) AUXILIARY;
  DECLARE line_control_reg BYTE AT (8000H + 3) AUXILIARY;
  DECLARE modem_control_reg BYTE AT (8000H + 4) AUXILIARY;
  DECLARE line_status_reg BYTE AT (8000H + 5) AUXILIARY;
  DECLARE modem_status_reg BYTE AT (8000H + 6) AUXILIARY;

DECLARE cr LITERALLY 'ODH';

DECLARE user_que_size LITERALLY '180';

DECLARE trans_user_buf( user_que_size ) BYTE AUXILIARY;
DECLARE trans_user_que_head BYTE;
DECLARE trans_user_que_tail BYTE;
DECLARE trans_user_que_count BYTE;
DECLARE trans_user_que_full BIT PUBLIC;
DECLARE trans_user_que_empty BIT PUBLIC;

DECLARE request_user_flow_off BIT;
DECLARE request_user_flow_on BIT;

DECLARE trans_user_flow_control BIT;

DECLARE recv_user_buf( user_que_size ) BYTE AUXILIARY;
DECLARE recv_user_que_head BYTE;
DECLARE recv_user_que_tail BYTE;
DECLARE recv_user_que_count BYTE;
DECLARE recv_user_que_full BIT;
DECLARE recv_user_que_empty BIT PUBLIC;

DECLARE recv_user_que_low_threshold LITERALLY 'user_que_size-10';

DECLARE sw1_image BYTE PUBLIC;
DECLARE sw2_image BYTE PUBLIC;
DECLARE sw3_image BYTE PUBLIC;

DECLARE first_escape_found BIT;
DECLARE session_escape BIT PUBLIC;

us enable: PROCEDURE PUBLIC USING O;

/* Procedure to initialize the User port interrupts. */

DISABLE;

interrupt_enable_reg = 0000001B;
EOC = 1;

ENABLE;
PL/M-61 COMPILER  USIUT - User Interface Layer Module

46 1 END user_enable;
47 2 user_disable: PROEDURE PUBLIC USING 0;
48 2 /*------------------------------------------*/
49 2 /* user_disable */
50 2 /* Procedure to disable the User port interrupts: */
51 2 DISABLE;
52 2 interrupt_enable_reg = 00000000B;
53 2 EX0 = 0;
54 2 ENABLE;
55 2 END user_disable;
56 2 user_queue_initialize: PROEDURE PUBLIC USING 0;
57 2 /*------------------------------------------*/
58 2 /* user_queue_initialize */
59 2 /* Procedure to initialize both the user transmit queue and the user */
60 2 /* receive queue. */
61 2 trans_user_queue_tail = user_queue_size - 1;
62 2 trans_user_queue_head = 0;
63 2 trans_user_queue_count = 0;
64 2 trans_user_queue_full = 0;
65 2 trans_user_queue_empty = 1;
66 2 recv_user_queue_tail = user_queue_size - 1;
67 2 recv_user_queue_head = 0;
68 2 recv_user_queue_count = 0;
69 2 recv_user_queue_full = 0;
70 2 recv_user_queue_empty = 1;
71 2 first_escape_found = 0;
72 2 session_escape = 0;
73 2 request_user_flow_off = 0;
74 2 request_user_flow_on = 0;
75 2 trans_user_flow_control = 0;
76 1 END user_queue_initialize;
77 2 user_get_switches: PROEDURE PUBLIC USING 0;
78 2 /*------------------------------------------*/
79 2 /* user_get_switches */
80 2 /* Procedure to read the switch settings of switch banks SW1, SW2, and SW3 */
81 2 /* into bytes sw1_image, sw2_image, and sw3_image respectively. A 1 bit */
82 2 /* indicates an open switch and a 0 bit indicates a closed switch. */
83 2 DECLARE p1_3 BIT AT( 003H ) REGISTER; /* Bit 3 of Port 1 */
84 2 DECLARE p1_4 BIT AT( 004H ) REGISTER; /* Bit 4 of Port 1 */
85 2 DECLARE p1_5 BIT AT( 005H ) REGISTER; /* Bit 5 of Port 1 */
86 2 DECLARE i BYTE;
PL/N-51 COMPILER  USRLAY - User Interface Layer Module

75 2   swi_image = 0;
76 2   sw2_image = 0;
77 2   sw3_image = 0;
78 3   DO i=0 to 7:
79 3     Pi = ( P1 AND 11000000B ) OR ( 00110000B OR i );
80 3     IF p1_3 THEN swi_image = swi_image + 1;
81 3     swi_image = ROR( swi_image, 1 );
82 3     IF p1_4 THEN sw2_image = sw2_image + 1;
83 3     sw2_image = ROR( sw2_image, 1 );
84 3     IF p1_5 THEN sw3_image = sw3_image + 1;
85 3     sw3_image = ROR( sw3_image, 1 );
86 3     END;
87 1   END user_get_switches;
88 2   user_set_baud: PROCEDURE( data ) PUBLIC USING 0;
89 2     --------------------------------------------------------
90 2     /* user_set_baud */
91 2     /* Procedure to get the baud rate of the 8250 User Serial port without */
92 2     /* affecting other line parameters. */
93 2     /* */
94 2     DECLARE data BYTE;
95 2     /* */
96 2     DECLARE rate_table( 8 ) WORD CONSTANT( 0000H, 0011H, 0040H, 0080H, 0180H, 0300H, 0417H, 0800H );
97 2     line_control_reg = line_control_reg OR 80H;  /* Select divisor registers */
98 2     data_reg = LOW( rate_table( data AND 07H ) );
99 2     interrupt_enable_reg = HIOR( rate_table data AND 07H );
100 2     line_control_reg = line_control_reg AND 7FH;  /* Select standard registers*/
101 1   END user_set_baud;
102 2   user_initialise: PROCEDURE( data ) PUBLIC USING 0;
103 2     ---------------------------------------------------------
104 2     /* user_initialise */
105 2     /* Procedure to initialize the 8250 User Serial Port based on the byte value */
106 2     /* passed. */

96
bits
3.2.1 Band Rate Number of stop bits
0 0 0 9600 1
0 0 1 4800 1
0 1 0 19200 1
0 1 1 12000 1
1 0 0 300 1
1 0 1 150 1
1 1 0 110 2
1 1 1 75 2

bit 4 0 = Parity disabled
     1 = Parity enabled

bit 5 0 = Even parity
     1 = Odd parity

bit 6 0 = 8 bit word length
     1 = 7 bit word length

bits 7, 8 No effect on the 8250

100 2 DECLARE data BYTE;
101 2 DECLARE p1_7 BIT AT (08H) REGISTER;  /* Bit 7 of Port 1 */

/* Reset the 8250 chip */
102 2 p1_7 = 1;
103 2 p1_7 = 0;
104 2 CALL par_set_baud( data );

/* Set parity enable/disable */
105 2 IF ( data AND 08H ) = 0
     THEN line_control_reg = ( line_control_reg AND 08H );
106 2 ELSE line_control_reg = ( line_control_reg OR 08H );

/* Set parity odd/even */
107 2 IF ( data AND 10H ) = 0
     THEN line_control_reg = ( line_control_reg OR 10H );
108 2 ELSE line_control_reg = ( line_control_reg AND OFEH );

/* Set data word length */
109 2 IF ( data AND 20H ) = 0
     THEN line_control_reg = ( line_control_reg OR 03H );
110 2 ELSE line_control_reg = ( line_control_reg AND OFEH ) OR 02H;

111 1 END usr_initialize;

112 2 usr_in_byte_status: PROCEDURE BIT PUBLIC USING 0;
113 2 IF ( line_status_reg AND 01H ) = 0
     THEN RETURN 0;
ELSE RETURN 1;

END user_in_byte_status;

user_in_byte: PROCEDURE BYTE PUBLIC USING 0;
FLICTURE-------------------------------------------------------------*/
/* user_in_byte
/* Taped procedure which returns a character when available
/ * from the User Serial Port receiver.

DO WHILE ( line_status_reg AND 01H ) = 0;
    CALL check_status;
    END;

RETURN data_reg;

END user_in_byte;

user_out_byte: PROCEDURE( data ) PUBLIC USING 0;
CLUDRCE---------------------------------------------------------------*/
/* user_out_byte
/* Procedure which sends a character out the User Serial Port.

DO WHILE ( line_status_reg AND 40H ) = 0; END;

data_reg = data;

END user_out_byte;

user_auto_baud: PROCEDURE PUBLIC USING 0;
CLUDRCE---------------------------------------------------------------*/
/* user_auto_baud
/* Procedure to set the baud rate of the 8250 User Serial port automatically
/* based on incoming carriage returns.

DECLARE data BYTE;

DO WHILE ( data_status_reg AND 00H ) = 0; END;

DECLARE 1 BYTE;
DECLARE junk BYTE;
DECLARE found BIT;

found = 0;

DO WHILE NOT found;

CALL user_set_baud( 1 );

buffer_timer = OFFH;

DO WHILE ( buffer_timer <> 0 ) AND ( NOT found ) AND ( i<8 );

CALL check_status;

IF user_in_byte_status
THEN IF data_reg = CR / * If found a carriage return then set */
    THEN DO; / * baud rate and exit */
        found = 1;
        sw1_image = ( sw1_image AND OF8H ) OR 1;
    END;
END;
PL/M-61 COHPILEB.

140 5 ELSE DO;
150 5 i = i + 1;
160 5 CALL time( 265 );
161 5 CALL time( 265 );
162 5 CALL time( 265 );
163 5 CALL time( 265 );
164 5 IF i<6 THEN CALL user_set_band( i );
165 5 junk = data_reg;
166 5 buffer_timer = OFFH;
167 5 END;
168 4 END;
180 3 END;
161 3 END user_auto_hand;
163 2 trans_user_que_enq: PROCEDURE( data ) PUBLIC USING 0;
164 2 DECLARE data BYTE;
165 2 IF NOT trans_user_que_full
166 2 /* If the queue is full on call, then do nothing; data is lost. */
167 2 /* If the queue is not full then:
168 2 THEN DO;
169 2 /* Enque the byte.
170 3 DISABLE;
171 3 trans_user_que_empty = 0;
172 3 trans_user_buf( trans_user_que_head ) = data;
173 3 trans_user_que_head = trans_user_que_head + 1;
174 3 IF trans_user_que_head = trans_user_que_size
175 3 THEN trans_user_que_head = 0;
176 3 trans_user_que_count = trans_user_que_count + 1;
177 3 IF trans_user_que_count = trans_user_que_size
178 3 THEN trans_user_que_full = 1;
179 3 /* Initiate packet transmission, if it has stopped */
180 3 IF ( ( interrupt_enable_reg AND OEH=0 ) AND NOT trans_user_flow_control
181 3 THEN interrupt_enable_reg = interrupt_enable_reg OR OEH;
182 3 ENABLE;
183 3 END;
180 1 END trans_user_que_enq;
181 2 recv_user_que_deq: PROCEDURE BYTE PUBLIC USING 0;
182 2 /* recv_user_que_deq */
DECLARE data BYTE;

IF recv_user_queue_empty
    /* If the queue is empty on call, then do nothing; return null data. */
    THEN RETURN 0;
ELSE
    /* If the queue is not empty then: */
    ELSE DO;
        /* Deque the byte. */
        DISABLE;
        recv_user_queue_tail = 0;
        recv_user_queue_tail = recv_user_queue_tail + 1;
        IF recv_user_queue tail = user_queue_size
            THEN recv_user_queue_tail = 0;
        recv_user_queue_count = recv_user_queue_count - 1;
        IF recv_user_queue_count = 0
            THEN recv_user_queue_empty = 1;
        data = recv_user_buff(recv_user_queue_tail);
        ENABLE;
        RETURN data;
END;
END recv_user_queue_deque;

PROCEDURE INDIRECTCALLABLE INTERRUPT 0;
/* Procedure to service interrupts from the User Port (the INS8260 ACE chip). */

DECLARE id BYTE;
DECLARE data BYTE;

/* Main code of ins8260_serv */

id = interrupt_id_reg;
DO WHILE (id AND DIR) = 0;
    DO CASE SRC( (id AND OSH), 1 );
    modem_status_serv: DO;
/*****/
PL/M-61 COMPILER   USRLAY - User Interface Layer Module

/ * modem_status_serv */
/ * Procedure to service modem control line interrupts. */
/ * */

206 4 END modem_status_serv;

207 5 chr_trans_serv: DO;
/ * chr_trans_serv */
/ * */

208 5 data = line_status_reg; /* Clear the status reg */

209 6 IF trans_usr_que_empty
   THEN interrupt_enable_reg = interrupt_enable_reg AND OFDH;
   ELSE
      trans_usr_que_deque: DO;
      /* trans_usr_que_deque */
      /* Procedure to dequeue a data byte from the user transmit queue. */
      /* If the queue is not empty then: */
      /* Dequeue the byte. */
      trans_usr_que_full = 0;
      trans_usr_que_tail = trans_usr_que_tail + 1;
      IF trans_usr_que_tail = trans_usr_que_size
         THEN trans_usr_que_tail = 0;
      trans_usr_que_count = trans_usr_que_count - 1;
      IF trans_usr_que_count = 0
         THEN trans_usr_que_empty = 1;
      data_reg = trans_usr_buf( trans_usr_que_tail );
   END trans_usr_que_deque;
   END chr_trans_serv;

212 4 END chr_trans_serv;

213 5 chr_recv_serv: DO;
/ * chr_recv_serv */
/ * */

214 6 recv_usr_que_enqueue: PROCEDURE( data );
/ * recv_usr_que_enqueue */
/ * Procedure to enqueue a data byte onto the user receive queue. */

215 6 DECLARE data BYTE;

216 6 IF NOT recv_usr_que_full
   /* If the queue is full on call, then do nothing; data is lost. */
/* If the queue is not full then: */
THEN DO;

/* Enque the byte. */
recv_user_que_empty = 0;
recv_user_buf( recv_user_que_head ) = data;
recv_user_que_head = recv_user_que_head + 1;
IF recv_user_que_head = user_que_size
THEN recv_user_que_head = 0;
recv_user_que_count = recv_user_que_count + 1;
IF recv_user_que_count = user_que_size
THEN DO;
recv_user_que_full = 1;
recv_user_que_overflow = 1;
END;

END recv_user_que_enque;

/* Main code */
data = line_status_reg;  /* Clear the status reg */
data = data_reg;  /* Get the received byte */

/* If enabled, check for escape sequence */
IF session_escape_state
THEN IF first_escape_found
THEN IF data = LOW( session_escape_sequence )
THEN DO;
first_escape_found = 0;
session_escape = 1;
END;
ELSE DD;
first_escape_found = 0;
CALL recv_user_que_enque( HIGH( session_escape_sequence ) );
CALL recv_user_que_enque( data );
END;
ELSE IF data = HIGH( session_escape_sequence )
THEN first_escape_found = 1;
ELSE CALL recv_user_que_enque( data );
ELSE CALL recv_user_que_enque( data );

END chr_recv_serv;

line_status_serv: DO;
/* Procedure to service line status interrupts. */
END line_status_serv:
END; /* END DO CASE */

id = interrupt_id_reg;

END; /* END DO WHILE */

END invaly;

MODULE INFORMATION:

<table>
<thead>
<tr>
<th>Description</th>
<th>(STATIC+OVERLAYABLE)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CODE SIZE</td>
<td>0310h 7848</td>
</tr>
<tr>
<td>CONSTANT SIZE</td>
<td>0010h 16D</td>
</tr>
<tr>
<td>DIRECT VARIABLE SIZE</td>
<td>00h+06h 50+ 60</td>
</tr>
<tr>
<td>INDIRECT VARIABLE SIZE</td>
<td>00h+00h 0D+ 0D</td>
</tr>
<tr>
<td>BIT SIZE</td>
<td>0Ah+01h 10D+ 1D</td>
</tr>
<tr>
<td>BIT-ADDRESSABLE SIZE</td>
<td>00h+00h 0D+ 00</td>
</tr>
<tr>
<td>AUXILIARY VARIABLE SIZE</td>
<td>0140h 3200</td>
</tr>
<tr>
<td>MAXIMUM STACK SIZE</td>
<td>0014h 200</td>
</tr>
<tr>
<td>REGISTER-BANK(S) USED:</td>
<td>0 2</td>
</tr>
<tr>
<td>636 LINES READ</td>
<td>0</td>
</tr>
</tbody>
</table>

END OF PL/M-51 COMPILATION
Appendix D
Transport Layer Module

PL/N-61 COMPILER TRLAY Transport Layer Module

TSIS-II PL/N-61 V1.1
COMPILED INVOKED BY: PLM61:F8:TRLAY.JNK

$PAGEWIDTH( 95 )
$TITLE('TRLAY Transport Layer Module')

DECLARE trlay: D0;
/--------------------------------------------------------------------------/
/*
/* trlay
/*
/* Dave Wickliff
/* Last Modified: 11/27/86
/*
/*

DECLARE signot LITERALLY
'crlf, 'ADSL LAN Version 1.0. Local Address = ');

$NULIST
/--------------------------------------------------------------------------/
/*
/* Externals from USRLAY
/*
6 1 DECLARE trans_usr_qos_full BIT EXTERNAL;
7 1 DECLARE recv_usr_qos_empty BIT EXTERNAL;
8 1 DECLARE sw1_image BYTE EXTERNAL;
9 1 DECLARE sw2_image BYTE EXTERNAL;
10 1 DECLARE sw3_image BYTE EXTERNAL;
11 1 DECLARE session_escape BIT EXTERNAL;

104
SLEEP
12 2 user_initialize: PROCEDURE( data ) EXTERNAL;
13 2 DECLARE data BYTE; END;
15 2 user_enable: PROCEDURE EXTERNAL; END;
17 2 user_disable: PROCEDURE EXTERNAL; END;
19 2 user_qe_initialize: PROCEDURE EXTERNAL; END;
21 2 user_set_switches: PROCEDURE EXTERNAL; END;
23 2 user_set_band: PROCEDURE( data ) EXTERNAL;
24 2 DECLARE data BYTE; END;
26 2 user_auto_band: PROCEDURE EXTERNAL; END;
28 2 trans_qe_enqe: PROCEDURE( data ) EXTERNAL;
29 2 DECLARE data BYTE; END;
31 2 recv_user_qe_deque: PROCEDURE BYTE EXTERNAL; END;

/* Externals from NETLAY */
33 2 net_initialize: PROCEDURE EXTERNAL; END;
35 2 trans_qe_enqe: PROCEDURE EXTERNAL; END;
37 1 DECLARE trans_qe_enqe WORD EXTERNAL;
38 1 DECLARE trans_qe_queue BIT EXTERNAL;
39 2 recv_qe_deque: PROCEDURE EXTERNAL; END;
41 1 DECLARE recv_qe_deque WORD EXTERNAL;
42 1 DECLARE recv_qe_queue BIT EXTERNAL;
43 1 DECLARE network_status BYTE EXTERNAL;
44 1 DECLARE session_active BIT EXTERNAL;

/* Externals from DLLAY */
46 1 DECLARE data_link_status BYTE EXTERNAL;

/* Externals from TIMSRV */
46 1 DECLARE buffer_timer BYTE EXTERNAL;
47 1 DECLARE buffer_timer_restart BYTE EXTERNAL;

/* Externals from PWTST */
48 2 power_on_test: PROCEDURE EXTERNAL; END;

/* Externals from COMLAY */
50 2 command_layer: PROCEDURE EXTERNAL; END;
52 2 message: PROCEDURE( ptr ) EXTERNAL;
53 2 DECLARE ptr ADDRESS;
54 2 END;

/* Externals from COMCOM */
55 2 display_hex: PROCEDURE( data ) EXTERNAL;
56 2 DECLARE data BYTE;
57 2 END;

/* Externals from DLXLAY */
68 1 DECLARE random BYTE EXTERNAL;

105
$INCLUDE(:f8:errmsg.dcl)

/* Error/Status messages for SETLAY */

DECLARE error_retry LITERALLY '01H';
DECLARE error_retry_mag LITERALLY '01H';
DECLARE error_watch_dog LITERALLY '02H';
DECLARE error_watch_dog_mag LITERALLY '02H';
DECLARE stat_session_via_remote LITERALLY '10H';
DECLARE stat_session_via_remote_mag LITERALLY '10H';
DECLARE stat_session_via_local LITERALLY '11H';
DECLARE stat_session_via_local_mag LITERALLY '11H';
DECLARE stat_remote_busy LITERALLY '12H';
DECLARE stat_remote_busy_mag LITERALLY '12H';
DECLARE stat_remote_not_available LITERALLY '13H';
DECLARE stat_remote_not_available_mag LITERALLY '13H';
DECLARE stat_close_via_local LITERALLY '14H';
DECLARE stat_close_via_local_mag LITERALLY '14H';
DECLARE stat_close_via_remote LITERALLY '15H';
DECLARE stat_close_via_remote_mag LITERALLY '15H';
DECLARE error_retry LITERALLY '02H';
DECLARE error_retry_mag LITERALLY '02H';
DECLARE error_jammed LITERALLY '02H';
DECLARE error_jammed_mag LITERALLY '02H';
DECLARE defer_timeout LITERALLY '03H';
DECLARE defer_timeout_mag LITERALLY '03H';

// Local Definitions */

DECLARE errstat_entry LABEL PUBLIC;
DECLARE loop_forever LABEL;
DECLARE command_state BIT PUBLIC;
DECLARE var_flow_state BIT PUBLIC;
DECLARE xon_xoff_flow_state BIT PUBLIC;
DECLARE verbose_state BIT PUBLIC;
DECLARE session_escape_state BIT PUBLIC;
DECLARE dtr_control_state BIT PUBLIC;
DECLARE auto_baud_state BIT PUBLIC;
DECLARE local_addr WORD PUBLIC;
DECLARE remote_addr WORD PUBLIC;
DECLARE session_escape_sequence WORD PUBLIC;
92 DECLARE crlf LITERALLY 'ODH,0AH';      /* Carriage Return, Line Feed */
93 DECLARE bell LITERALLY '0TH';          /* Bell */
94 DECLARE mem_begin LITERALLY '0000H';   /* Start of External Memory */
96 DECLARE mem_end   LITERALLY '03FFH';    /* End of External Memory */
98 init: PROCEDURE;
    /*-----------------------------------------------*/
    /* init */
    /* System initialization procedure. */
    /* Power on states defined by the three DIP switch settings as follows: */
    /* Note Switch Open = 1 and Switch Close = 0 */

DIP SWITCH 1:

bits 3,2,1  Baud Rate  Number of stop bits
  0  0  0   9600      1
  0  0  1   4800      1
  0  1  0  1800      1
  0  1  1  1200      1
  1  0  0   300       1
  1  0  1   150       1
  1  1  0   110       2
  1  1  1    76       2

bit 4  0 = Parity disabled
      1 = Parity enabled

bit 5  0 = Even parity
      1 = Odd parity

bit 6  0 = 8 bit word length
      1 = 7 bit word length

bit 7  0 = Flow control enabled
      1 = Flow control disabled

bit 8  0 = XON/XOFF flow control
      1 = EIA flow control

DIP SWITCH 2:

bit 1-8 Low order byte of node address

DIP SWITCH 3:

bits 1-4 Low nibble of high order byte of node address
      Note the high nibble is assumed 0000

bit 6  0 = Verbose enabled
      1 = Verbose disabled
bit 5  0 = Session escape enabled  
       1 = Session escape disabled  

bit 7  0 = DTR control disabled  
       1 = DTR control enabled  

bit 8  0 = Autobaud disabled  
       1 = Autobaud enabled  

07 2  CALL user_get_switches;  
08 2  CALL user_initialize(sw1_image);  
09 2  local_addr = SHL(DOUBLE(sw3_image AND OFN), 8 ) * sw2_image;  
100 2  sw1_image = ROL(sw1_image, 1 );  
101 2  xon_xoff_flow_state = NOT BOOLEAN(sw1_image);  
102 2  sw1_image = ROL(sw1_image, 1 );  
103 2  user_flow_state = NOT BOOLEAN(sw1_image);  
104 2  sw1_image = ROR(sw1_image, 2 );  
105 2  sw3_image = ROL(sw3_image, 1 );  
106 2  auto_baud_state = BOOLEAN(sw3_image);  
107 2  sw3_image = ROL(sw3_image, 1 );  
108 2  dtr_control_state = BOOLEAN(sw3_image);  
109 2  sw3_image = ROL(sw3_image, 1 );  
110 2  session_escape_state = NOT BOOLEAN(sw3_image);  
111 2  sw3_image = ROL(sw3_image, 1 );  
112 2  verbose_state = NOT BOOLEAN(sw3_image);  
113 2  sw3_image = ROR(sw3_image, 4 );  
114 2  command_state = 1;  
115 2  buffer_timer_restart = 6H;  
116 2  session_escape_sequence = 18H;  /* Control-X Control-X */  
117 2  CALL net_initialize;  
118 1  END init;  
119 2  _check_errstat: PROCEDURE PUBLIC;  
       /*-----------------------------------------------*/  
       /*_check_errstat */  
       /*-----------------------------------------------*/  
120 2  IF ( network_status <> 0 ) OR ( data_link_status <> 0 )  
       THEN GOTO errstat_entry;  
122 1  END check_errstat;
display_errorstat: PROCEDURE;
/*---------------------------------------------*/
/* display_errorstat */
/*---------------------------------------------*/

IF ( network_status <> O ) AND ( network_status <> stat_session_via_remote )
AND ( network_status <> stat_close_via_remote )
THEN DO:
    CALL message( .( crlf, '!', bell, 'NET'. O ) );
    CALL display_hex( network_status );
    IF verbose_state
THEN DO:
        CALL message( .( ' - ', O ) );
        IF network_status = error_retry
THEN CALL message( .( error_retry_msg, O ) );
        ELSE IF network_status = error_watch_dog
THEN CALL message( .( error_watch_dog_msg, O ) );
        ELSE IF network_status = stat_session_via_local
THEN CALL message( .( stat_session_via_local_msg, O ) );
        ELSE IF network_status = stat_remote_busy
THEN CALL message( .( stat_remote_busy_msg, O ) );
        ELSE IF network_status = stat_remote_not_available
THEN CALL message( .( stat_remote_not_available_msg, O ) );
        ELSE IF network_status = stat_close_via_local
THEN CALL message( .( stat_close_via_local_msg, O ) );
        END;

    CALL message( .( crlf, O ) );
END:

network_status = O;

IF data_link_status <> O
THEN DO:
    CALL message( .( crlf, '!', bell, 'DLK'. O ) );
    CALL display_hex( data_link_status );
    IF verbose_state
THEN DO:
        CALL message( .( ' - ', O ) );
        IF data_link_status = error_backoff
THEN CALL message( .( error_backoff_msg, O ) );
        ELSE IF data_link_status = error_jammed
THEN CALL message( .( error_jammed_msg, O ) );
        ELSE IF data_link_status = error_defer
THEN CALL message( .( error_defer_msg, O ) );
        END;

    CALL message( .( crlf, O ) );
END;

data_link_status = O;

END display_errorstat;

talk: PROCEDURE;
/*---------------------------------------------*/
/* talk */
/*---------------------------------------------*/
DECL. TRAN. BLOCK LENGTH BASED TRAN. PKT. QUE. IN BYTE AUXILIARY;

DECLARE TRAN. BLOCK PTR WORD;

DECLARE TRAN. BLOCK BASED TRAN. BLOCK PTR BYTE AUXILIARY;

DECLARE REC. BLOCK LENGTH BASED REC. PKT. QUE. OUT BYTE AUXILIARY;

DECLARE REC. BLOCK PTR WORD;

DECLARE REC. BLOCK BASED REC. BLOCK PTR BYTE AUXILIARY;

TRAN. BLOCK LENGTH = 0;

TRAN. BLOCK PTR = TRAN. PKT. QUE. IN + 1;

REC. BLOCK LENGTH = 0;

REC. BLOCK PTR = REC. PKT. QUE. OUT + 1;

COMMAND STATE = 0;

CALL USER. QUE. INITIALIZE;

CALL USER. ENABLE;

DO WHILE NOT COMMAND STATE;

    RANDOM = RANDOM + 1;

    IF SESSION. ESCAPE THEN COMMAND STATE = 1;

    CALL CHECK. ERR. STAT;

    DO WHILE ( NOT REC. USER. QUE. EMPTY ) AND ( TRAN. BLOCK LENGTH <= 122 );

        TRAN. BLOCK = REC. USER. QUE. DEQUE;

        TRAN. BLOCK LENGTH = TRAN. BLOCK LENGTH + 1;

        TRAN. BLOCK PTR = TRAN. BLOCK PTR + 1;

        BUFFER. TIMER = BUFFER. TIMER. RESTART;

    END;

    IF ( BUFFER. TIMER = 0 ) OR ( TRAN. BLOCK LENGTH = 122 )

    THEN IF ( NOT TRAN. PKT. QUE. FULL ) AND ( TRAN. BLOCK LENGTH > 0 )

    THEN DO;

        CALL TRAN. PKT. QUE. ENQUE;

        TRAN. BLOCK LENGTH = 0;

        TRAN. BLOCK PTR = TRAN. PKT. QUE. IN + 1;

    END;

    DO WHILE ( NOT TRAN. USER. QUE. EMPTY ) AND ( REC. BLOCK LENGTH < 0 );

        CALL TRAN. USER. QUE. ENQUE( REC. BLOCK );

        REC. BLOCK LENGTH = REC. BLOCK LENGTH - 1;

        REC. BLOCK PTR = REC. BLOCK PTR + 1;

    END;

    IF ( NOT REC. PKT. QUE. EMPTY ) AND ( REC. BLOCK LENGTH = 0 )

    THEN DO;

        CALL REC. PKT. QUE. DEQUE;

        REC. BLOCK PTR = REC. PKT. QUE. OUT + 1;

    END;

END;
CALL user_disable;

END talk;

/*---------------------------------------------------------------*/
/* Main Code                                                   */
/*---------------------------------------------------------------*/

CALL power_on_test;

CALL init;

IF NOT auto_band_state
    THEN DO:
        CALL message(( signon, 0 ));
        CALL display_hex(HIGH(local_addr));
        CALL display_hex(LOW(local_addr));
    END;

loop_forever::;

IF auto_band_state AND ( NOT session_active )
    THEN DO:
        CALL user_auto_band;
        CALL message(( signon, 0 ));
        CALL display_hex(HIGH(local_addr));
        CALL display_hex(LOW(local_addr));
    END;

IF command_state OR ( NOT session_active )
    THEN CALL command_layer;

errstat_entry:;

CALL user_disable;

CALL display_errstat;

IF session_active THEN CALL talk;

GOTO loop_forever; /* End forever loop */

END trlay;

MODULE INFORMATION:

<table>
<thead>
<tr>
<th>CODE SIZE</th>
<th>0336H</th>
<th>821D</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONSTANT SIZE</td>
<td>018FH</td>
<td>361D</td>
</tr>
<tr>
<td>DIRECT VARIABLE SIZE</td>
<td>08H+04H</td>
<td>6D+ 4D</td>
</tr>
<tr>
<td>INDIRECT VARIABLE SIZE</td>
<td>0CH+0CH</td>
<td>OD+ OD</td>
</tr>
<tr>
<td>BIT SIZE</td>
<td>07H+00H</td>
<td>7D+ OD</td>
</tr>
<tr>
<td>BIT-ADDRESSABLE SIZE</td>
<td>00H+00H</td>
<td>OD+ 0D</td>
</tr>
<tr>
<td>AUXILIARY VARIABLE SIZE</td>
<td>0000H</td>
<td>0D</td>
</tr>
<tr>
<td>MAXIMUM STACK SIZE</td>
<td>0000H</td>
<td>8D</td>
</tr>
</tbody>
</table>

495 LINES READ
0 PROGRAM ERROR(S)

END OF PL/N-61 COMPILATION
Appendix E

Command Layer Module

PL/M-61 Compiler  COMLAY Command Layer Module

ISIS-II PL/M-61 V1.1
COMPILER INVOKED BY: PLM61 :F8:COMLAY.JNK

$PAOEWIDTH( 06 )
$TITLE('COMLAY Command Layer Module')

comlay: DO;
/*---------------------------------------------*/
/* comlay */
/*
/* Dave Wickliff */
/* Last Modified: 11/27/86 */
/*
/*---------------------------------------------*/

/* Externals from USRLAY */
DECLARE swl_image BYTE EXTERNAL;
DECLARE sw2_image BYTE EXTERNAL;
DECLARE sw3_image BYTE EXTERNAL;
DECLARE user_initialize: PROCEDURE( data ) EXTERNAL;
DECLARE user_get_switches: PROCEDURE EXTERNAL; END;
DECLARE user_auto_band: PROCEDURE EXTERNAL; END;
DECLARE user_In_byte: PROCEDURE BYTE EXTERNAL; END;
DECLARE user_out_byte: PROCEDURE( data ) EXTERNAL;
DECLARE Declare data BYTE; END;

/* Externals from COMCOM */
DECLARE tree( 17 ) STRUCTURE( command( 11 ) BYTE,
para_ptr WORD ) EXTERNAL CONSTANT;
DECLARE verbose_state BIT EXTERNAL;
DECLARE command_state BIT EXTERNAL;
DECLARE address_command: PROCEDURE EXTERNAL; END;
DECLARE autobaud_command: PROCEDURE EXTERNAL; END;
DECLARE baud_command: PROCEDURE EXTERNAL; END;
DECLARE connect_command: PROCEDURE EXTERNAL; END;
DECLARE disconnect_command: PROCEDURE EXTERNAL; END;
DECLARE start_command: PROCEDURE EXTERNAL; END;
DECLARE escape_command: PROCEDURE EXTERNAL; END;
DECLARE flow_command: PROCEDURE EXTERNAL; END;
DECLARE help_command: PROCEDURE EXTERNAL; END;
DECLARE parity_command: PROCEDURE EXTERNAL; END;
DECLARE resume_command: PROCEDURE EXTERNAL; END;
DECLARE status_command: PROCEDURE EXTERNAL; END;
DECLARE time_command: PROCEDURE EXTERNAL; END;
DECLARE verbose_command: PROCEDURE EXTERNAL; END;
DECLARE word_command: PROCEDURE EXTERNAL; END;
DECLARE crlf LITERALLY '0DH,0AH'; /* Carriage Return, Line Feed */
DECLARE bell LITERALLY '07H'; /* Bell */
DECLARE be LITERALLY '08H'; /* Backspace */
DECLARE del LITERALLY '07H'; /* Delete */
DECLARE evaluate BIT; /* Parameter of 'get_string' */
DECLARE fill BIT; /* Parameter of 'get_string' */
DECLARE question BIT; /* Parameter of 'get_string' */
DECLARE match_end BYTE; /* Parameter of 'compare' */
DECLARE text_ptr BYTE;
DECLARE string_end BYTE;
DECLARE command_number BYTE;
DECLARE parameter_number BYTE PUBLIC;
DECLARE hex_parameter WORD PUBLIC;
DECLARE string(30) BYTE AUXILIARY;
DECLARE str_p ADDRESS;
DECLARE char BASED str_p BYTE CONSTANT;
DO WHILE char <> 0;
    CALL var_out_byte( char );
    str_p = str_p + 1;
END;
END message;
PL/M-61 COMPILER  COMMAND Layer Module

74 2 print_string: PROCEDURE;
    /*-----------------------------*/
    /* print_string */
    /* */
    /* Procedure to print a prompt and the current text string. */
    /* */
76 2 DECLARE i BYTE;
76 2 CALL message(.(crlf, ':', 0));
77 2 i = 0;
78 3 DO WHILE i <> string_end;
79 3 CALL user_out_byte( string( i ));
80 3 i = i + 1;
81 3 END;
82 1 END print_string;
83 2 print_confirmation_help: PROCEDURE;
    /*-----------------------------*/
    /* print_confirmation_help */
    /* */
    /* Procedure to print help information on available confirmations. */
    /* */
84 2 IF verbose_state
    THEN DO;
88 3 CALL message(.( '?', crlf, crlf, 'Confirm command with carriage return.'
    crlf, 0 ));
87 3 CALL print_string;
88 3 END;
89 3 ELSE DO;
90 3 CALL user_out_byte( '?' );
91 3 string( string_end ) = '?';
92 3 string_end = string_end + 1;
93 3 END;
94 1 END print_confirmation_help;
95 2 print_parameter: PROCEDURE( ptr );
    /*-----------------------------*/
    /* print_parameter */
    /* */
    /* Procedure to print a list of parameters pointed to by ptr. */
    /* */
96 2 DECLARE i BYTE;
97 2 DECLARE ptr WORD;
98 2 DECLARE char BASED ptr BYTE CONSTANT;
99 2 IF char <> 0
    THEN CALL message(.( '(', 0 ));
101 2 ELSE CALL message(.( crlf, 0 ));
102 3 DO WHILE char <> 0;
CALL user_out_byte( char );
ptr = ptr + 1;
i = i + 1;
END;
}

ptr = ptr + 10 - i;

IF char <> 0
THEN CALL message( ( ' ' | 0 ));
ELSE CALL message( ( ' ' | crlf | 0 ));
END;

END print_parameter;

print_parameter_help: PROCEDURE( ptr );
/*--------------------*/
/* Procedure to print help information on available parameters. */
/*--------------------*/
DECLARE ptr WORD;
DECLARE char BASED ptr BYTE CONSTANT;

IF verbose_state
THEN DO;
  IF char = 0
  THEN CALL print_confirmation_help;
  ELSE DO;
    CALL message( ( ' ? ' | crlf | crlf | ' Available parameters --' | crlf | 0 ));
    CALL message( ( ' ' | 0 ));
    CALL print_parameter( ptr );
    CALL print_string;
  END;
  ELSE DO;
    CALL user_out_byte( '?' );
    string( string_end ) = ' '; string_end = string_end + 1;
  END;
END print_parameter_help;

print_command: PROCEDURE PUBLIC;
/*----------------------*/
DECLARE I BYTE;
DECLARE J BYTE;

CALL message( ( crlf, crlf, 'Available commands and their parameters -', crlf, 0 ) );
i = 0;
DO WHILE tree( i ).command( 0 ) <> 0:
    CALL message( ( ' ', ' ', 0 ) );
    j = 0;
    DO WHILE tree( i ).command( j ) <> 0:
        CALL user_out_byte( tree( i ).command( j ) );
        j = j + 1;
    END;
    CALL print_parameter( tree( i ).para_ptr );
    i = i + 1;
END;

END print_command;

print_command_help: PROCEDURE;
/*----------------------------------------------*/
/* print_command_help */
/*----------------------------------------------*/
/* Procedure to print help information on available commands. */

IF verbose_state
    THEN DO;
        CALL message( ( ' ? ', 0 ) );
        CALL print_command;
    END;
        CALL print_string;
    END;
ELSE DO;
    CALL user_out_byte( ' ? ' );
    string( string_end ) = ' ? ';
    string_end = string_end + 1;
    END;
END;

print_command_syntax_error: PROCEDURE;
/*----------------*/
/* print_command_syntax_error */
/*----------------*/
PL/N-SI Compiler  CONLAY Command Layer Module

175 2 CALL message(. ( crlf, '!', bell, 'CONCE', 0 ));
176 2 IF verbose_state THEN CALL message(. ( ' - Command Syntax Error.', 0 ));
177 2 CALL message(. ( crlf, crlf, '!', 0 ));
179 2 string_end = 0;
180 1 END print_command_syntax_error;
181 2 print_parameter_syntax_error: PROCEDURE;
182 2 IF verbose_state THEN CALL message(. ( ' - Parameter Syntax Error..', 0 ));
183 2 CALL message(. ( crlf, crlf, '!', 0 ));
184 2 string_end = 0;
187 1 END print_parameter_syntax_error;
188 2 print_fill_error: PROCEDURE;
189 2 IF verbose_state THEN CALL message(. ( bell, '!', 0 ));
192 2 string( string_end ) = ' ';
193 2 string_end = string_end + 1;
194 1 END print_fill_error;
195 2 print_fill: PROCEDURE( ptr );
196 2 DECLARE ptr WORD;
197 2 DECLARE char BASED ptr BYTE CONSTANT;
198 2 IF verbose_state THEN DO;
200 4 DO WHILE char <> 0;
201 4 CALL user_out_byte( char );
202 4 string( string_end ) = char;
203 4 string_end = string_end + 1;
204 4 text_ptr = text_ptr + 1;
205 4 ptr = ptr + 1;
208 4 END;
207 3 END;
209 2 get_string: PROCEDURE;
209 2 DECLARE ptr WORD;
209 2 DECLARE char BASED ptr BYTE CONSTANT;
DECLARE char BYTE;

DO WHILE (char <> cr) AND (char <> ' ') AND (char <> '?');

IF (char = bs) OR (char = del)
THEN DO;

IF string_end <> 0
THEN DO;

CALL message("bs", "", bs, 0);

string_end = string_end - 1;

END;

END;

ELSE DO;

CALL user_out_byte(char);

string(string_end) = char;

string_end = string_end + 1;

END;

char = user_in_byte;

END;

evaluate = 0;

fill = 0;

question = 0;

IF char = cr
THEN evaluate = 1;

ELSE IF char = ' '
THEN fill = 1;

ELSE question = 1;

END get_string;

skip_space: PROCEDURE;
/*
* skip_space
*/

DO WHILE (text_ptr<string_end) AND (string(text_ptr)=' ');

text_ptr = text_ptr + 1;

END;

END skip_space;

compare: PROCEDURE(ptr) BYTE;
/*
* compare
*/

/*
* Procedure to compare the text of array string() starting at text_ptr
* with a constant command/parameter string pointed to by ptr. It
*
/*  returns a non-zero value if there is a unique match. The value is an offset to the non-significant remainder of the constant string. */
/*  Zero offset is returned with the bit ambiguous set if the text is ambiguous or with the bit ambiguous reset if the text is invalid. */

243 2 DECLARE t_ptr WORD;
244 2 DECLARE ptr WORD;
245 2 DECLARE char BASED ptr BYTE CONSTANT;
246 2 DECLARE offset ptr BYTE;
247 2 DECLARE hex_string_error BIT;

248 3 upper: PROCEDURE( ascii ) BYTE;
249 3 DECLARE ascii BYTE;
250 3 IF ( ascii >= 'a' ) AND ( ascii <= 'z' )
251 3 THEN RETURN ( ascii - 'a' );
252 3 ELSE RETURN ascii;
253 2 END upper;
254 3 hex_string: PROCEDURE;
255 3 DECLARE ( i, j, temp ) BYTE;
256 3 /* Skip to the end of the hex digits */
257 4 i = 0;
258 4 DO WHILE ( ( t_ptr <> '' ) AND ( t_ptr <> string_end ) );
259 4 i = i + 1;
260 4 t_ptr = t_ptr + 1;
261 3 END;
262 3 IF ( i = 0 ) OR ( i > 4 )
263 4 /* If the digit string is null or too short, then error. */
264 4 /* Convert the digit string */
265 4 ELSE DO;
266 4 hex_parameter = 0;
267 4 hex_string_error = 0;
268 4 /* Convert one digit at a time, from high order to low order */
269 5 DO j = t_ptr - i TO t_ptr - 1;

119
temp = string( );

IF ( temp > '8' ) AND ( temp < 'G' )
   THEN hex parameter = SHL( hex parameter, 4 ) OR ( temp - 'A' + 10
   - 3);
ELSE IF ( temp > ' ' ) AND ( temp < 'g' )
   THEN hex parameter = SHL( hex parameter, 4 ) OR ( temp - 'a' + 10
   - 3);
ELSE IF ( temp > '/' ) AND ( temp < ':' )
   THEN hex parameter = SHL( hex parameter, 4 ) OR ( temp - 'O' );
ELSE hex_string_error = 1;
END;
END;

END hex_string;

/-----------------------------------------------------------------------------*/
/* Start of main code for compare */
ambiguous = 0;
t_ptr = text_ptr;

IF char = 0
   /* If there is no constant string then: */
   THEN RETURN 0;
ELSE IF char < 10
   /* If the parameter should be a hex value, read and check the text */
   THEN DO;
   CALL hex_string;
   IF hex_string_error
      THEN RETURN 0;
   ELSE IF char = 2
      THEN IF hex parameter > 265
         THEN RETURN 0;
   ELSE DO;
      text_ptr = t_ptr;
      RETURN 1;
   END;
   ELSE DO;
      text_ptr = t_ptr;
      RETURN 1;
   END;
ELSE DO;
   /* If the constant is a text string then compare each character */
   offset = 0;
   DO WHILE ( upper( char ) = upper( string( t_ptr ) ) ) AND ( char <> 0 )
      AND ( string_end <> t_ptr );
      ptr = ptr + 1;
   offset = offset + 1;
   t_ptr = t_ptr + 1;
   END;

120
/* Determine what kind of match: unique, ambiguous, or invalid */

305 4 END;
306 3 IF char = 'O'
307 4 THEN DO:
308 4 text_ptr = t_ptr;
309 4 RETURN offset;
310 4 END;

311 3 ELSE IF string_end = t_ptr
312 4 THEN IF ( char >= 'a' )
313 4 THEN DO:
314 4 text_ptr = t_ptr;
315 4 RETURN offset;
316 4 END;
317 3 ELSE RETURN 0;

318 3 ELSE IF char >= 'a' and string( t_ptr ) == '
319 4 THEN DO:
320 4 text_ptr = t_ptr;
321 4 RETURN offset;
322 4 END;
323 4 ELSE DO:
324 4 ambiguous = 1;
325 4 RETURN 0;
326 4 END;
327 3 END;

328 1 END compare;

329 2 parse_confirmation: PROCEDURE( ptr ) BIT:
330 2 DECLARE ptr WORD;

331 2 /* Skip any leading spaces */
332 2 CALL skip_space;

333 2 IF text_ptr=string_end
334 2 /* If there is no text in the string, then: */
335 2 THEN IF question /* Print parameter help for the command */
336 3 THEN DO:
337 3 CALL print_confirmation_help;
338 3 RETURN 0;
339 3 END;
340 2 ELSE IF fill
341 2 /* Or put the space in the string */
342 2 THEN DO: /* Or put the space in the string */
343 3 CALL nmr_out_byte( ' ' );
344 3 string( string_end ) = ' ';
345 3 string_end = string_end + 1;
346 3 RETURN 0;
BEGIN:

/* Else if there is remaining text */
ELSE IF fill
    THEN CALL print_fill_error;
    ELSE IF evaluate
        THEN CALL print_parameter_syntax_error;
    ELSE CALL print_parameter_help( ptr );
    RETURN 0;
END; /* ELSE DO */

END parse_confirmation;

parse_parameter: PROCEDURE ( ptr ) BIT;
  /* parse_parameter */
  /* A typed procedure to search a parameter tree pointed to by ptr for */
  /* parameters that match array string( ). If a valid parameter is found, */
  /* then it returns set and parameter_number indicates the matched */
  /* parameter. */
  DECLARE ptr WORD;
  DECLARE char BASED ptr BYTE CONSTANT;
  DECLARE found_parameter BIT;
  DECLARE para_tree WORD;
  para_tree = ptr;
  /* Skip any leading spaces */
  CALL skip_space;
  IF text_ptr=string_end
    THEN IF question /* Print parameter help for the command */
        THEN DO;
        CALL print_parameter_help( para_tree );
        RETURN 0;
        END;
    ELSE IF fill
        THEN DO; /* Or put the space in the string */
            CALL char_oct_byte( ' ' );
            string( string_end ) = ' ';
            string_end = string_end + 1;
            RETURN 0;
        END;
    ELSE IF char = 0 /* Or end the line */
        THEN RETURN 1; /* Done if no parameters */
    ELSE DO;
        CALL print_parameter_syntax_error; /* Else error */
        RETURN 0;
  END;
END;

122
/* Else parse the remaining text */
381 3  ELSE DO;
382 3  found_parameter = 0;
383 3  parameter_number = 0;
384 4  DO WHILE ( NOT found_parameter ) AND ( char <> 0 );
385 4      match_end = compare( ptr );
386 4      IF match_end <> 0
387 5      THEN found_parameter = 1;
388 5      ELSE DO;
389 6      parameter_number = parameter_number + 1;
390 6      ptr = ptr + 10;
391 6      END;
392 4      END;
393 3  IF found_parameter
394 4      THEN DO;
395 5      CALL print_fill( ptr + match_end );
396 5      RETURN parse_confirmation( para_tree );
397 5      END;
398 4  ELSE DO;
399 5  IF fill
400 6      THEN CALL print_fill_error;
401 6      ELSE IF evaluate
402 7      THEN CALL print_parameter_syntax_error;
403 6      ELSE CALL print_parameter_help( para_tree );
404 5      RETURN 0;
405 4      END;
406 3  END;  /* ELSE DO */
407 1  END parse_parameter;
408 2  parse_command: PROCEDURE BIT;
409 2  DECLARE found_command BIT;
410 2  /* Skip any leading spaces */
411 2  text_ptr = 0;
412 2  CALL skip_space;
413 2  IF text_ptr=string_end
414 3  /* If there is no text in the string, then: */
415 4  THEN IF question /* Print full command help */
416 3  CALL print_command_help;
417 3  RETURN 0;
PL/M-61 COMPILER  CONCAT Command Layer Module.

417  END;
418 ELSE IF fill
419 THEN DO; /* Or put the space in the string */
420 CALL user_out_byte( ' ' );
421 string( string_end ) = ' ';
422 string_end = string_end + 1;
423 RETURN 0;
424 END;
425 ELSE DO; /* Or end the line */
426 CALL message( (.crlf, ':', 0 ) );
427 string_end = 0;
428 RETURN 0;
429 END;

/* Else parse the remaining text */
430 ELSE DO;
431 found_command = 0;
432 command_number = 0;
433 DO WHILE ( NOT found_command ) AND ( tree( command_number ).command( 0 ) <> 0 );
434   match_end = compare( .tree( command_number ).command( 0 ) );
435 IF match_end <> 0 THEN found_command = 1;
436 ELSE command_number = command_number + 1;
437 END;
438 IF found_command THEN DO;
439 CALL print_fill( .tree( command_number ).command( match_end ) );
440 RETURN parse_parameter( .tree( command_number ).para_ptr );
441 END;
442 ELSE DO;
443 IF fill
444 THEN CALL print_fill_error;
445 ELSE IF evaluate
446 THEN CALL print_command_syntax_error;
447 ELSE CALL print_command_help;
448 RETURN 0;
449 END;
450 END;
451 END; /* ELSE DO */
452 END parse_command;
453
454 command_layer: PROCEDURE PUBLIC;
455 /* command_layer */
456 command_state = 1;
457 CALL message( (.crlf, ':', 0 ) );
458 string_end = 0;
459 DO WHILE command_state;

-124-
CALL get_string;

IF parse_command
    THEN DO;
        DO CASE command_number;
            CALL address_command;
            CALL autobaud_command;
            CALL baud_command;
            CALL connect_command;
            CALL disconnect_command;
            CALL dtr_command;
            CALL escape_command;
            CALL flow_command;
            CALL help_command;
            CALL parity_command;
            CALL resume_command;
            CALL status_command;
            CALL time_command;
            CALL vterm_command;
            CALL word_command;
        END;
    IF command_state THEN DO;
        CALL message('(. crlf, ":",
            O )');
        string_end = O;
    END;
    END;
END;

END command_layer;

END conlay;

MODULE INFORMATION: (STATIC+OVERLAYABLE)

<table>
<thead>
<tr>
<th>Type</th>
<th>Code Size</th>
<th>Direct Variable Size</th>
<th>Indirect Variable Size</th>
<th>Bit Size</th>
<th>Bit-Addressable Size</th>
<th>Auxiliary Variable Size</th>
<th>Stack Size</th>
<th>Error(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code Size</td>
<td>0764H</td>
<td>078+0DH</td>
<td>00H+00H</td>
<td>04H+03H</td>
<td>00H+00H</td>
<td>001EH</td>
<td>0012H</td>
<td>0</td>
</tr>
<tr>
<td>Constant Size</td>
<td>010EH</td>
<td>270D</td>
<td>7D+13D</td>
<td>0D</td>
<td>0D</td>
<td>30D</td>
<td>18D</td>
<td>0</td>
</tr>
</tbody>
</table>

END OF PL/M-61 COMPILATION
Appendix F

Command Layer Command Routines Module

PL/N-61 COMPILER  CONCOM Command Layer Command Routines Module

ISIS-II PL/N-61 V.1.1
COMPILER INVOKED BY: PLM61 :FS:COMCOM.JNK

$PAG])]TR( 95 )
$TITLE('CONCOM Command Layer Command Routines Module')

1 1  comom: DO;
/* comom
/*
/* Dave Wickliff
/* Last Modified: 11/27/85
/*
*/

/* Externals from USRLAY */
1 /*
2 1 DECLARE swl_image BYTE EXTERNAL;
3 1 DECLARE sw2_image BYTE EXTERNAL;
4 1 DECLARE sw3_image BYTE EXTERNAL;
5 2 usrl_set_bytes: PROCEDURE( data ) EXTERNAL;
6 2 DECLARE data BYTE END;
7 2 usrl_initialize: PROCEDURE( data ) EXTERNAL;
8 2 DECLARE data BYTE END;
9 2 usrl_set_bytes: PROCEDURE BYTE EXTERNAL END;
10 2 DECLARE byte BYTE END;
11 2 DECLARE data BYTE END;
12 2 DECLARE data BYTE END;

/* Externals from COMLAY */
13 /*
14 2 message: PROCEDURE( ptr ) EXTERNAL;
15 2 DECLARE ptr WORD END;
16 2 print_command: PROCEDURE EXTERNAL END;
17 1 DECLARE parameter_number BYTE EXTERNAL;
18 1 DECLARE hex_parameter WORD EXTERNAL;
$EXCEPT$

/------------------------------------------------------------------------/

/* Externals from NETLAY */

session_request_open: PROCEDURE EXTERNAL; END;
session_request_close: PROCEDURE EXTERNAL; END;

DECLARE network_status BYTE EXTERNAL;
DECLARE session_active BIT EXTERNAL;
DECLARE total_valid_recv WORD;
DECLARE total_invalid_recv WORD;
DECLARE total_retries WORD;
DECLARE peak_retries BYTE;

/------------------------------------------------------------------------/

/* Externals from DLKAY */

DECLARE data_link_status BYTE EXTERNAL;
DECLARE total_backoffs WORD;
DECLARE peak_backoffs BYTE;
DECLARE total_defers WORD;
DECLARE peak_defers BYTE;

/------------------------------------------------------------------------/

/* Externals from TIMSER */

DECLARE buffer_timer_restart BYTE EXTERNAL;

/------------------------------------------------------------------------/

/* Externals from TLKAY */

DECLARE command_state BIT EXTERNAL;
DECLARE err_flow_state BIT EXTERNAL;
DECLARE non_xoff_flow_state BIT EXTERNAL;
DECLARE xoff_xon_state BIT EXTERNAL;
DECLARE session_escape_state BIT EXTERNAL;
DECLARE dtr_control_state BIT EXTERNAL;
DECLARE auto_baud_state BIT EXTERNAL;

DECLARE local_addr WORD EXTERNAL;
DECLARE remote_addr WORD EXTERNAL;
DECLARE session_escape_sequence WORD EXTERNAL;

/* Parameter table structure: 

For each table, there are a series of 10 byte fields with a null field 
at the end of the table.

A field consists of either:

The left justified ASCII text of a textual parameter 
padded with null bytes.  { For commands with textual 
parameters }

Or

Null bytes.  { For parameterless commands and table terminators }
Or

The first byte is a value indicating the maximum number of hex digits in the parameter, and is padded with null bytes. (For commands with numeric parameters)

```
DECLARE para_address( * ) BYTE CONSTANT
/* '1234567890' */
4,0,0,0,0,0,0,0,0,0,0,0,0;

DECLARE para_autohand( * ) BYTE CONSTANT
/* '1234567890' */
'Enable' , 0,0,0,0,
'Disable' , 0,0,0,

DECLARE para_baud( * ) BYTE CONSTANT
/* '1234567890' */
'75' , 0,0,0,0,0,0,0,0,
'110' , 0,0,0,0,0,0,0,0,
'150' , 0,0,0,0,0,0,0,0,
'300' , 0,0,0,0,0,0,0,0,
'1200' , 0,0,0,0,0,0,0,0,
'1800' , 0,0,0,0,0,0,0,0,
'4800' , 0,0,0,0,0,0,0,0,
'9600' , 0,0,0,0,0,0,0,0,

DECLARE para_connect( * ) BYTE CONSTANT
/* '1234567890' */
4,0,0,0,0,0,0,0,0,0,0,0,0,

DECLARE para_disconnect( * ) BYTE CONSTANT
/* '1234567890' */
0;

DECLARE para_dtr( * ) BYTE CONSTANT
/* '1234567890' */
'Enable' , 0,0,0,0,
'Disable' , 0,0,0,

DECLARE para_esc( * ) BYTE CONSTANT
/* '1234567890' */
'Disable' , 0,0,0,
4,0,0,0,0,0,0,0,0,0,0,0,0,0;

DECLARE para_flow( * ) BYTE CONSTANT
/* '1234567890' */
'Acn/xoff' , 0,0,
'Re232' , 0,0,0,0,0,0,
'Disable' , 0,0,0,0;
```

128
PL/1 Compiler    CONCOM Command Layer Command Routines Module

49 DECLARE para_help( ) BYTE CONSTANT(
   /= '1234567890' /*
       0 */;
   0 );

50 DECLARE para_parity( ) BYTE CONSTANT(
   /= '1234567890' /*
       'Odd' 0,0,0,0,0,0,
       'Even' 0,0,0,0,0,0,
       'Disable' 0,0,0,
       0 */;
   0 );

51 DECLARE para_resume( ) BYTE CONSTANT(
   /= '1234567890' /*
       0 */;
   0 );

52 DECLARE para_status( ) BYTE CONSTANT(
   /= '1234567890' /*
       0 */;
   0 );

53 DECLARE para_time( ) BYTE CONSTANT(
   /= '1234567890' /*
       0 */;
   0 );

54 DECLARE para_verbose( ) BYTE CONSTANT(
   /= '1234567890' /*
       'Enable' 0,0,0,0,
       'Disable' 0,0,0,
       0 */;
   0 );

55 DECLARE para_word( ) BYTE CONSTANT(
   /= '1234567890' /*
       '8bit' 0,0,0,0,0,0,
       '7bit' 0,0,0,0,0,0,
       0 */;
   0 );

   /* Command table */

56 DECLARE tru (17) STRUCTURE(
   command( 11 ) BYTE,
   para_ptr WORD
 ) PUBLIC CONSTANT(
   /= '1234567890' /*
       'Address' 0,0,0,0,
       'Autobaud' 0,0,0,
       'Band' 0,0,0,0,0,0,
       'Connect' 0,0,0,0,
       'Disconnect' 0,
       'Dtr' 0,0,0,0,0,0,0,
       'Escape' 0,0,0,0,0,
       'Flow' 0,0,0,0,0,0,
       'Help' 0,0,0,0,0,0,
       'Parity' 0,0,0,0,0,
       'Pseudo' 0,0,0,0,0,
       'Status' 0,0,0,0,0,
       'Timeout' 0,0,0,0,
       0 );
DECLARE crlf LITERALLY 'ODH, OAH'; /* Carriage Return, Line Feed */
DECLARE bell LITERALLY '07H'; /* Bell */
DECLARE cr LITERALLY '0DH'; /* Carriage Return */
DECLARE bs LITERALLY '08H'; /* Backspace */
DECLARE del LITERALLY '7FH'; /* Delete */

DECLARE (d1, d2) BYTE;

temp = SHR(data, 4);
IF temp > 9 THEN CALL user_out_byte(temp + 'A' - 10);
ELSE CALL user_out_byte(temp + '0');

IF temp > 11 THEN CALL user_out_byte(temp + 'A' - 10);
ELSE CALL user_out_byte(temp + '0');

END display_hex;

address_command: PROCEDURE PUBLIC;
/* ------------------------------- */
/* address_command */
/* */

IF session_active
THEN CALL message((crlf, 'j', bell, 'Error - Can not change local address', crlf, 0));
ELSE local_addr = hex_parameter;
END address_command;

autobaud_command: PROCEDURE PUBLIC;
/* ------------------------------- */
/* autobaud_command */
/* */

IF parameter_number = 0
THEN auto_baud_state = 1;
ELSE auto_baud_state = 0;
END autobaud_command;

baud_command: PROCEDURE PUBLIC;
/* ------------------------------- */
/* baud_command */
/* */
CALL time( 266 ); /* Allow time for characters to leave the INS 8280 */
CALL time( 266 );
CALL time( 266 );
CALL time( 266 );

swi_image = ( swi_image AND OFSH ) OR ( NOT parameter_number ) AND OTH;
CALL user_set_baud( swi_image );

END baud_command;

connect_command: PROCEDURE PUBLIC;
/*--------------------------------------------------------------*/
/* connect_command */
/* */
remote_addr = hex_parameter;

CALL session_request_open;
DO WHILE (network_status = 0) AND (data_link_status = 0); END;
command_state = 0;
END connect_command;

disconnect_command: PROCEDURE PUBLIC;
/*--------------------------------------------------------------*/
/* disconnect_command */
/* */
IF session_active THEN CALL session_request_close;
END disconnect_command;

dtr_command: PROCEDURE PUBLIC;
/*--------------------------------------------------------------*/
/* dtr_command */
/* */
IF parameter_number = 0 THEN dtr_control_state = 1;
ELSE dtr_control_state = 0;
END dtr_command;

escape_command: PROCEDURE PUBLIC;
/*--------------------------------------------------------------*/
/* escape_command */
/* */
IF parameter_number = 0 THEN session_escape_state = 0;
ELSE DO;
session_escape_state = 1;
session_escape_sequence = hex_parameter;
END;
IF session_active THEN DO;
    CALL message(.(crlf, 0));
    command_state = 0;
END;
END resume_command;

status_command: PROCEDURE PUBLIC;

display_state: PROCEDURE( state );

DECLARE state BIT:

IF state THEN CALL message(.( 'Enabled', 0 ));
ELSE CALL message(.( 'Disabled', 0 ));
END display_state;

display_baud: PROCEDURE( rate );

DECLARE rate BYTE;

CALL message( .para_baud( 10 * rate ));
END display_baud;

CALL message(.( crlf, crlf, ' Node Status = ', crlf,
    Local Node Address = ', 0 ));
CALL display_hex( HIGH( local_addr ) );
CALL display_hex( LOW( local_addr ) );
CALL message(.( crlf,
    ' Autobaud Mode = ', 0 ));
CALL display_state( auto_baud_state );
CALL message(.( crlf,
    ' Baud Rate = ', 0 ));
CALL display_baud( ( NOT ewi_image) AND 0TH );
CALL message(.( crlf,
    ' DTR Control = ', 0 ));
CALL display_state( dtr_control_state );
CALL message(.( crlf,
    ' Connect Escape = ', 0 ));
CALL display_state( session_escape_state );
PL/M-61 COMPILER  CONCOM Command Layer Command Routines Module

114 1 END escape_command;
115 2 flow_command: PROCEDURE PUBLIC;
116 2 /*-------------------------------*/
117 2 /* flow_command */
118 2 /* */
119 2 IF parameter_number = 0
120 3 THEN D0;
121 3 xon_xoff_flow_state = 1;
122 3 usr_flow_state = 1;
123 2 END IF parameter_number = 1
124 3 THEN D0;
125 3 xon_xoff_flow_state = 0;
126 3 usr_flow_state = 1;
127 2 ELSE usr_flow_state = 0;
128 1 END flow_command;
129 2 help_command: PROCEDURE PUBLIC;
130 2 /*-------------------------------*/
131 2 /* help_command */
132 2 /* */
133 2 CALL print_command;
134 2 CALL message(( crlf, 0 ));
135 1 END help_command;
136 2 parity_command: PROCEDURE PUBLIC;
137 2 /*-------------------------------*/
138 2 /* parity_command */
139 2 /* */
140 2 IF parameter_number = 0
141 3 THEN swi_image = swi_image OR 18H;
142 3 ELSE IF parameter_number = 1
143 3 THEN swi_image = ( swi_image AND OETH ) OR O8H;
144 2 ELSE swi_image = swi_image AND O8H;
145 2 CALL time( 256 );  /* Allow time for characters to leave the INS 8250 */
146 2 CALL time( 256 );  
147 2 CALL time( 256 );  
148 2 CALL time( 256 );
149 2 CALL time( 256 );
150 2 CALLusr_initialise( swi_image );
151 1 END parity_command;
152 2 resume_command: PROCEDURE PUBLIC;
153 2 /*-------------------------------*/
154 2 /* resume_command */
155 2 /* */
156 2
157 132
IF session_active
THEN DO;
    CALL message( ( crlf, 0 ));
    command_state = 0;
END;

END resume_command;

status_command: PROCEDURE PUBLIC;

display_state: PROCEDURE( state );

DECLARE state BIT;

IF state
    THEN CALL message( ( 'Enabled', 0 ));
    ELSE CALL message( ( 'Disabled', 0 ));

END display_state;

display_band: PROCEDURE( rate );

DECLARE rate BYTE;

CALL message( .para_band( 10 = 'Rate' ) );

END display_band;

CALL message( ( crlf, crlf, ' Node Status --', crlf,
    ' Local Node Address --', 0 ));
CALL display_hex( HIGH( local_addr ) );
CALL display_hex( LOW( local_addr ) );
CALL message( ( crlf,
    ' Autobaud Mode --', 0 ));
CALL display_state( auto_baud_state );
CALL message( ( crlf,
    ' Baud Rate --', 0 ));
CALL display_band( ( NOT swl_image ) AND OTH );
CALL message( ( crlf,
    ' DTR Control --', 0 ));
CALL display_state( dtr_control_state );
CALL message( ( crlf,
    ' Connect Escape --', 0 ));
CALL display_state( session_escape_state );
173 2. IF session_escape_state
   THEN DO:
   176 3.   CALL message( "", 0 );
   176 3.   CALL display_hex( HIGH( session_escape_sequence ) );
   177 3.   CALL display_hex( LOW( session_escape_sequence ) );
   178 3.   END;
   176 2. CALL message( "crlf,
   180 2.   ' Flow Control - '', 0 );
   IF user_flow_state
   THEN IF xon_xoff_flow_state
   THEN CALL message( 'XON/XOFF', 0 );
   ELSE CALL message( 'RS-232', 0 );
   ELSE CALL message( 'Disabled', 0 );
   185 2. CALL message( "crlf,
   186 2.   ' Parity - '', 0 );
   IF ( aw1_image AND 00H ) <> 0
   THEN IF ( aw1_image AND 10H ) <> 0
   THEN CALL message( 'Odd', 0 );
   ELSE CALL message( 'Even', 0 );
   ELSE CALL message( 'Disabled', 0 );
   191 2. CALL message( "crlf,
   192 2.   ' Buffer timeout - '', 0 );
   CALL display_hex( buffer_timer_restart );
   193 2. CALL message( "crlf,
   194 2.   ' Verbose Mode - '', 0 );
   CALL display_hex( verbose_status );
   196 2. CALL message( "crlf,
   196 2.   ' Word Length - '', 0 );
   IF ( aw1_image AND 20H ) <> 0
   THEN CALL message( '7 bit', 0 );
   ELSE CALL message( '8 bit', 0 );
   199 2. CALL message( "crlf,
   200 2.   ' Current Session - '', 0 );
   IF session_active
   THEN DO:
   202 3.   CALL message( 'Active with remote node', 0 );
   203 3.   CALL display_hex( HIGH( remote_addr ) );
   204 3.   CALL display_hex( LOW( remote_addr ) );
   205 3.   END;
   206 2. ELSE CALL message( 'No session active', 0 );
   207 2. CALL message( "crlf, 0 );
   208 1. END status_command;
   209 2. time_command: PROCEDURE PUBLIC;
   /*-----------------------------------------------*/
   /* time_command */
   */

134
buffer_timer_restart = LOW( hex_parameter );

END time_command;

verbatim_command: PROCEDURE PUBLIC;
/*----------------------------------------*/
/* verbatim_command */
/*----------------------------------------*/

IF parameter_number = 0 THEN verbose_state = 1;
ELSE verbose_state = 0;
END verbose_command;

verbatim_command: PROCEDURE PUBLIC;
/*----------------------------------------*/
/* word_command */
/*----------------------------------------*/

IF parameter_number = 0 THEN swi_image = swi_image AND ODFH;
ELSE swi_image = swi_image OR 20H;

CALL time( 265 ); /* Allow time for characters to leave the INS 8260 */

CALL time( 265 );
CALL time( 265 );
CALL time( 265 );
CALL time( 265 );

CALL swr_initialize( swi_image );

END word_command;

END comcom;

MODULE INFORMATION: (STATIC+OVERLAYABLE)
CODE SIZE = 02FH 763D
CONSTANT SIZE = 040H 1037D
DIRECT VARIABLE SIZE = 00H+02H 0D+ 2D
INDIRECT VARIABLE SIZE = 00H+00H 0D+ 0D
BIT SIZE = 00H+01H 0D+ 1D
BIT-ADDRESSABLE SIZE = 00H+00H 0D+ 0D
AUXILIARY VARIABLE SIZE = 0000H 0D
MAXIMUM STACK SIZE = 0004H 4D
REGISTER-BANK(S) USED: 0
524 LINES READ
0 PROGRAM ERROR(S)
END OF PL/N-51 COMPILATION
Appendix G
Network Layer Module

PL/M-61 Compiler  NEILAY Network Layer Module

ISIS-IT PL/M-61 V1.1
COMPILER INVOKED BY: PLMS1 :FB :NEILAY.JMK

$SPACEWIDTH( 96 )
$TITLE( "NEILAY Network Layer Module" )
$REGISTEBANK( 1 )

1 1 netlay: DD:
_FUNCTION
/* D&VI Wickliff */
/*
* Last Modified: 11/27/86
*/
$H Ol IS $INST
$FUNCTION

/• Externals from TALAY */
5 1 DECLARE local_addr WORD EXTERNAL;
6 1. DECLARE remote_addr WORD EXTERNAL;
7 1 DECLARE nsp_status_lad BIT AT( 98H ) REGISTER;

/• Externals from DLK1AY */
8 1 DECLARE data_link_status BYTE EXTERNAL;
9 1 DECLARE packet_trans_wait BIT EXTERNAL;
10 2 DATA_INITIALIZE: PROCEDURE EXTERNAL USING 0; END;
$\text{DECL}$

\text{//------------------------------/}
\text{// Externals from timer_initial: PROCEDURE EXTERNAL USING 0; END;}
\text{// timer variables and restart values}
\text{//------------------------------/}
\text{DECL watch_dog_timer BYTE EXTERNAL;}
\text{DECL watch_dog_timer_restart LITERALLY '00H';}

\text{//-------------------------------/}
\text{// Local Declarations}
\text{//-------------------------------/}
\text{DECL control_data LITERALLY '00000000';}
\text{DECL control_open_session LITERALLY '00000000';}
\text{DECL control_close_session LITERALLY '00000000';}
\text{DECL control_flow_off LITERALLY '00000000';}
\text{DECL control_watch_dog LITERALLY '00000000';}

\text{// Acknowledgment Types}
\text{//-------------------------------/}
\text{DECL ack_ok LITERALLY '00000000';}
\text{DECL ack_not_available LITERALLY '00000000';}
\text{DECL ack_busy LITERALLY '00000000';}

\text{// Position of the flow control field in an acknowledgement}
\text{DECL ack_flow_field LITERALLY '00000000';}

\text{DECL max_retry_count LITERALLY '01H';}
\text{DECL trans_data_pkt_buf(4) STRUCTURE(}
\text{\hspace{1cm}destination_address WORD,}
\text{\hspace{1cm}source_address WORD,}
\text{\hspace{1cm}control BYTE,}
\text{\hspace{1cm}data_length BYTE,}
\text{\hspace{1cm}data(122) BYTE,}
\text{\hspace{1cm}check_sum BYTE)} AUXILIARY;}

\text{DECL trans_pkt_que_head BYTE;}
\text{DECL trans_pkt_que_tail BYTE;}
\text{DECL trans_pkt_que_in WORD PUBLIC;}
\text{DECL trans_pkt_que_out WORD PUBLIC;}
\text{DECL trans_pkt_que_full BIT PUBLIC;}
\text{DECL trans_pkt_que_empty BIT;}

\text{DECL recv_data_pkt_buf(8) STRUCTURE(}
\text{\hspace{1cm}destination_address WORD,}
\text{\hspace{1cm}source_address WORD,}
\text{\hspace{1cm}control BYTE,}
\text{\hspace{1cm}data_length BYTE,}
\text{\hspace{1cm}data(122) BYTE,}
\text{\hspace{1cm}check_sum BYTE)} AUXILIARY;}

\text{DECL recv_pkt_que_head BYTE;}
\text{DECL recv_pkt_que_tail BYTE;}
\text{DECL recv_pkt_que_in WORD PUBLIC;}
\text{DECL recv_pkt_que_out WORD PUBLIC;}

137
DECLARE recv_pkt_queue_full BIT;
DECLARE recv_pkt_queue_empty BIT PUBLIC;
/* Number of packets in recv before recv 'flow off' */
DECLARE recv_pkt_queue_low_threshold LITERALLY '00H';

DECLARE control_open_pkt_buf STRUCTURE(
    destination_address WORD,
    source_address WORD,
    control BYTE,
    data_length BYTE,
    check_sum BYTE
) AUXILIARY;

DECLARE control_close_pkt_buf STRUCTURE(
    destination_address WORD,
    source_address WORD,
    control BYTE,
    data_length BYTE,
    check_sum BYTE
) AUXILIARY;

DECLARE control_flow_off_pkt_buf STRUCTURE(
    destination_address WORD,
    source_address WORD,
    control BYTE,
    data_length BYTE,
    check_sum BYTE
) AUXILIARY;

DECLARE control_watch_dog_pkt_buf STRUCTURE(
    destination_address WORD,
    source_address WORD,
    control BYTE,
    data_length BYTE,
    check_sum BYTE
) AUXILIARY;

/* Control packet queue flags */
DECLARE request_open_session BIT;
DECLARE request_close_session BIT;
DECLARE request_flow_off BIT;
DECLARE request_watch_dog BIT;

/* Acknowledgement evaluation state flags */
DECLARE pending_data BIT;
DECLARE pending_open_session BIT;
DECLARE pending_close_session BIT;
DECLARE pending_flow_off BIT;
DECLARE pending_watch_dog BIT;

DECLARE network_status BYTE PUBLIC;

$INCLUDE( ::8:errmsg.dcl )
/* Error/Status messages for NETLAY */
DECLARE error_retry LITERALLY '01H';

138
DECLARE error_retry_msg LITERALLY "Retry exceeded, no response from remote.");
DECLARE error_watch_dog LITERALLY "Watch dog time out, no response from remote.");
DECLARE stat_session_via_remote LITERALLY "Session established via remote.");
DECLARE stat_session_via_local LITERALLY "Session established via local.");
DECLARE stat_close_via_remote LITERALLY "Session closed via remote.");
DECLARE stat_close_via_local LITERALLY "Session closed via local.");
DECLARE stat_remote_busy LITERALLY "Remote is busy.");
DECLARE stat_remote_not_available LITERALLY "Remote is not available.");
DECLARE stat_close_not_available LITERALLY "Session closed not available.");
DECLARE close_retry_msg LITERALLY "Close retry exceeded.");
DECLARE error_jammed LITERALLY "Network jammed.");
DECLARE error_defer LITERALLY "Deferral timeout.");
DECLARE error_backoff LITERALLY "Backoff retry exceeded.");
DECLARE error_bkoff_msg LITERALLY "Backoff retry exceeded.");
DECLARE session_active BIT PUBLIC;
DECLARE recv_ack BYTE PUBLIC;
DECLARE trans_ack BYTE PUBLIC;
DECLARE retry_count BYTE;
DECLARE trans_sequence BYTE;
DECLARE recv_sequence BYTE;

DECLARE receive_flow_control BIT;

DECLARE transmit_flow_control BIT;

que_initialize: PROCEDURE PUBLIC USING 0;

/* Procedure to initialize both the data transmit queue and the data receive queue, as well as the control packet transmit 'queue'. */
PL/M-51 COMPILER  NETLAY Network Layer Module

/ * Globally callable using register bank 0.*/
100 trans_pkt_que_tail = 0;
101 trans_pkt_que_head = 1;
102 trans_pkt_que_in = trans_data_pkt_buf(0).data_length;
103 trans_pkt_que_full = 0;
104 trans_pkt_que_empty = 1;
105 recv_pkt_que_tail = 0;
106 recv_pkt_que_head = 1;
107 recv_pkt_que_out = recv_data_pkt_buf(0).data_length;
108 recv_pkt_que_in = recv_data_pkt_buf(1)
109 request_open_session = 0;
110 request_close_session = 0;
111 request_flow_off = 0;
112 request_watch_dog = 0;
113 END que_initialize;
114
115 que_initialize_local: PROCEDURE INDIRECTCALLABLE;
116
/ * Procedure to initialize both the data transmit queues and the data receive
queue, as well as the control packet transmit 'queue' */
117
118 trans_pkt_que_tail = 0;
119 trans_pkt_que_head = 1;
120 trans_pkt_que_in = trans_data_pkt_buf(0).data_length;
121 trans_pkt_que_full = 0;
122 trans_pkt_que_empty = 1;
123 recv_pkt_que_tail = 0;
124 recv_pkt_que_head = 1;
125 recv_pkt_que_out = recv_data_pkt_buf(0).data_length;
126 recv_pkt_que_in = recv_data_pkt_buf(1);
127 request_open_session = 0;
128 request_close_session = 0;
129 request_flow_off = 0;
130 request_watch_dog = 0;
131
132 net_initialize: PROCEDURE PUBLIC USING 0;
133
/ * Procedure to initialize the Network Layer and layers below it. */
DISABLE;

CALL data_initialize;  /* Initialize the Data Link Layer */
CALL timer_initialize;  /* Initialize the timer server */
CALL queue_initialize;  /* Initialize the receive and transmit queues */

session_active = 0;
user_status_led = 0;
pending_data = 0;
pending_open_session = 0;
pending_close_session = 0;
pending_flow_off = 0;
pending_watch_dog = 0;

ENABLE;

END net_initialize;

session_request_open: PROCEDURE PUBLIC USING 0;
/*-----------------------------------------------*/
/* session_request_open */
/* Procedure to initiate a request for a session. */
/*-----------------------------------------------*/

/* Build a Open Session packet and send */
control_open_pkt_buf.destination_address = remote_addr;
control_open_pkt_buf.source_address = local_addr;
control_open_pkt_buf.control = control_open_session;
control_open_pkt_buf.data_length = 0;
request_open_session = 1;
IF packet_trans_wait /* Restart transmitter if stopped. THEN EXI = 1; */

END session_request_open;

session_request_close: PROCEDURE PUBLIC USING 0;
/*-----------------------------------------------*/
/* session_request_close */
/* Procedure to initiate a request for session termination. */
/*-----------------------------------------------*/

/* Build a close Session packet and send */
control_close_pkt_buf.destination_address = remote_addr;
control_close_pkt_buf.source_address = local_addr;
control_close_pkt_buf.control = control_close_session;
control_close_pkt_buf.data_length = 0;
request_close_session = 1;
IF packet_trans_wait /* Restart transmitter if stopped. THEN EXI = 1; */

END session_request_close;

session_open: PROCEDURE INDIRECTLY_CALLABLE;
/*-----------------------------------------------*/
/* session_open */
PROCEDURE to start a session.

recv_sequence = 0;
trans_sequence = 0;
CALL que_initialize_local; /* Init the receive and transmit queues */
recv_flow_control = 0;
trans_flow_control = 0;
session_active = 1;
usr_status_led = 1;

END session_open;

session_close: PROCEDURE INDIRECTLY_CALLABLE:

Procedure to end a session.

session_active = 0;
usr_status_led = 0;

END session_close;

watch_dog_serv: PROCEDURE INDIRECTLY_CALLABLE PUBLIC:

Procedure to service a watch dog timeout. Called by Timer 0 interrupt.

IF session_active
THEN DO:
    /* Build and send a Watch_Dog packet */
    control_watch_dog_pkt_buf.destination_address = remote_addr;
    control_watch_dog_pkt_buf.source_address = local_addr;
    control_watch_dog_pkt_buf.control = control_watch_dog;
    control_watch_dog_pkt_buf.data_length = 0;
    request_watch_dog = 1;
    IF packet_trans_wait
    THEN EXI = 1;
END;

END watch_dog_serv;

trans_pkt_queue: PROCEDURE PUBLIC USING 0;

Procedure to enqueue the a data block onto the net transmit queue.

IF trans_pkt_queue_full
/* If the queue is full when called, then do nothing; data is lost */
THEN trans_pkt_queue_in = .trans_data_pkt_buf( trans_pkt_queue_head - 1 )
PL/NET-51 Compiler

/* If the queue is not full then: */
ELSE DO;
/* Encapsulate data block (build packet). */
trans_data_pkt_buf(trans_pkt_que_head).destination_address = remote_addr;
trans_data_pkt_buf(trans_pkt_que_head).source_address = local_addr;
trans_data_pkt_buf(trans_pkt_que_head).control = control_data OR trans_seq;
/* trans_sequence = (trans_sequence + 1) AND OXH; */
ELSE DO;
/* Enqueue the packet. */
DISABLE;
trans_pkt_que_empty = 0;
trans_pkt_que_head = (trans_pkt_que_head + 1) AND OXH;
trans_pkt_que_in = trans_data_pkt_buf(trans_pkt_que_head).data_length;
IF ((trans_pkt_que_head + 1) AND OXH) = trans_pkt_que_tail
THEN trans_pkt_que_full = 1;
/* Initiate packet transmission, if it has stopped */
IF packet_trans_wait AND NOT trans_flow_control
THEN EXI = 1;
ENABLE;
END;
END trans_pkt_que_enqueue;

trans_pkt_que_dequeue: PROCEDURE BIT INDIRECTLY_CALLABLE PUBLIC;
/*---------------------------------------------------------------*/
/* trans_pkt_que_dequeue */
/* Called procedure to dequeue a packet from the net transmit queue. When */
/* called recv_ack should contain the acknowledgement of the last packet */
/* dequeued, if applicable. The procedure returns set, if packet available*/
/* and trans_pkt_que_out points to the packet just dequeued. Otherwise, it */
/* returns reset, indicating no packets available and the transmitter */
/* should shut down. */
DECLARE retry BIT;
DECLARE error BIT;
/* If there is a error from the data link layer, then close session */
IF data_link_status <> 0
THEN error = 1;
/* If there is a pending acknowledgement, evaluate it: */
/* Evaluate acknowledgement for data packet. */
ELSE IF pending_data
THEN DO;
IF (recv_ack AND NOT ack_flow_field)
THEN DO;
IF (recv_ack AND ack_flow_field) <> 0
THEN trans_flow_control = 1;

143
208 4 pending_data = 0;
209 4 error = 0;
210 4 retry = 0;
211 4 retry_count = 0;
212 4 watch_dog_timer = watch_dog_timer_restart;
213 4 END;
214 3 ELSE IF retry_count < max_retry_count
215 4 THEN DO; /* Retransmit the packet. */
216 4 retry_count = retry_count + 1;
217 4 retry = 1;
218 4 error = 0;
219 4 END;
220 4 ELSE DO;
221 4 pending_data = 0;
222 4 network_status = error_retry;
223 4 error = 1;
224 4 END;
225 3 END;

// Evaluate acknowledgement for flow control packet. /*
226 2 ELSE IF pending_flow_off
227 3 THEN DO;
228 3 IF (recv_ack) = ack_ok
229 4 THEN DO;
230 4 recv_flow_control = 0;
231 4 pending_flow_off = 0;
232 4 error = 0;
233 4 retry = 0;
234 4 retry_count = 0;
235 4 watch_dog_timer = watch_dog_timer_restart;
236 4 END;
237 3 ELSE IF retry_count < max_retry_count
238 4 THEN DO; /* Retransmit the packet. */
239 4 retry_count = retry_count + 1;
240 4 retry = 1;
241 4 END;
242 4 ELSE DO;
243 4 pending_flow_off = 0;
244 4 network_status = error_retry;
245 4 error = 1;
246 4 END;
247 3 END;

// Evaluate acknowledgement for open session control packet. /*
248 2 ELSE IF pending_open_session
249 3 THEN DO;
250 3 IF (recv_ack) = ack_ok
251 4 THEN DO;
252 4 pending_open_session = 0;
253 4 error = 0;
254 4 retry = 0;
255 4 retry_count = 0;
256 4 network_status = stat_session_via_local;
257 4 CALL session_open;
watch_dog_timer = watch_dog_timer_restart;
END;
ELSE IF recv_ack = nack_busy
THEN DO;
pending_open_session = 0;
error = 1;
network_status = stat_remote_busy;
END;
ELSE IF recv_ack = nack_not_available
THEN DO;
pending_open_session = 0;
error = 1;
network_status = stat_remote_not_available;
END;
ELSE IF retry_count < max_retry_count
THEN DO; /* Retransmit the packet.
retry_count = retry_count + 1;
retry = 1;
END;
ELSE DO;
pending_open_session = 0;
network_status = error_retry;
END;
END:

/* Evaluate acknowledgement for close session control packet. */
ELSE IF pending_close_session
THEN DO:
 IF (recv_ack) = ack_ok
THEN DO;
pending_close_session = 0;
error = 1; /* Send no more packets and close session. */
retry_count = 0;
network_status = stat_close_via_local;
watch_dog_timer = watch_dog_timer_restart;
END;
ELSE IF retry_count < max_retry_count
THEN DO; /* Retransmit the packet.
retry_count = retry_count + 1;
retry = 1;
END;
ELSE DO;
pending_close_session = 0;
network_status = error_retry;
error = 1;
END;
END;
END;
/* Evaluate acknowledgement for watch dog control packet. */

305  2 ELSE IF pending_watch_dog

306  3 THEN DO:

307  3 IF ( recv_ack ) = ack_ok

308  4 THEN DO:

309  4 pending_watch_dog = 0;

310  4 error = 0;

311  4 retry = 0;

312  4 retry_count = 0;

313  4 watch_dog_timer = watch_dog_timer_restart;

314  4 END;

315  3 ELSE IF retry_count < max_retry_count

316  4 THEN DO: /* Retransmit the packet. */

317  4 retry_count = retry_count + 1;

318  4 retry = 1;

319  4 END;

320  4 ELSE DO;

321  4 pending_watch_dog = 0;

322  4 error = 1;

323  4 network_status = error_watch_dog;

324  4 END;

325  3 END;

326  3 ELSE DO;

327  3 retry_count = 0;

328  3 error = 0;

329  3 retry = 0;

330  3 END;

331  2 IF error

332  3 /* If there has been an error then close the session. */

333  3 THEN DO:

334  3 CALL session_close;

335  3 RETURN 0;

336  3 END;

337  2 ELSE IF retry

338  2 THEN RETURN 1;

339  2 ELSE IF request_open_session

340  2 THEN DO;

341  3 request_open_session = 0;

342  3 trans_pkt_que_out = control_open_pkt_buf;

343  3 pending_open_session = 1;

344  3 RETURN 1;

345  2 ELSE IF request_close_session

346  2 THEN DO;

347  3 request_close_session = 0;

348  3 trans_pkt_que_out = control_close_pkt_buf;

349  3 pending_close_session = 1;

350  3 RETURN 1;

351  2 END;

146
ELSE IF request_flow_off
    THEN DO;
        request_flow_off = 0;
        trans_pkt_que_out = .control_flow_off_pkt_buf;
        pending_flow_off = 1;
        RETURN 1;
        END;
    ELSE IF request_watch_dog
        THEN DO;
            request_watch_dog = 0;
            trans_pkt_que_out = .control_watch_dog_pkt_buf;
            pending_watch_dog = 1;
            RETURN 1;
            END;
    ELSE IF trans_flow_control
        THEN RETURN 0;
        ELSE DO;
            retry = NOT trans_pkt_QUE_empty;
            pending_data = retry;
            deque: DO;
            /*------------------------------------------------------*/
            /* deque */
            IF trans_pkt_QUE_empty
                /* If the queue is empty on call, then do nothing; return null pointer. */
                THEN trans_pkt_QUE_out = 0;
                IF the queue is not empty, then dequeue a packet.
                ELSE DO;
                    trans_pkt_QUE_tail = 0;
                    trans_pkt_QUE_tail = ( trans_pkt_QUE_tail + 1 ) AND 03H;
                    trans_pkt_QUE_out = .trans_data_pkt_buf( trans_pkt_QUE_tail );
                    IF ( ( trans_pkt_QUE_tail + 1 ) AND 03H ) = trans_pkt_QUE_head
                        THEN trans_pkt_QUE_empty = 1;
                        END;
                    END;
                    END deque;
                    RETURN retry;
                    END;
    END trans_pkt_QUE_deque;
recv_pkt_QUE_enque: PROCEDURE BIT INDIRECTCALLABLE PUBLIC;
/*------------------------------------------------------------*/
/* recv_pkt_QUE_enque */
/* Typed procedure to enqueue the current packet on to the net receive queue. On return recv_pkt_QUE_in points to a free packet to be used on the next enqueue. The procedure returns set if the received packet is to be acknowledged and trans_ack contains the acknowledgement */

DECLUE ok BIT; /* Evaluate the packet */
ok = 0;
trans_ack = recv_data_pkt_buf( recv_pkt_que_head ).control AND 1111110B;
IF trans_ack = control_data THEN DO;

IF session_active
THEN IF recv_data_pkt_buf( recv_pkt_que_head ).destination_address = local_addr
THEN IF recv_data_pkt_buf( recv_pkt_que_head ).source_address = remote_addr
THEN IF recv_data_pkt_buf( recv_pkt_que_head ).control = recv_sequence
THEN DO;

enque: DO;

IF recv_pkt_que_full /* If the queue is full when called, then do nothing; data is lost. */
THEN DO;
recv_pkt_que_in = recv_data_pkt_buf( recv_pkt_que_head - 1 ) AND 07H;
ok = 0;
END;

ELSE IF recv_pkt_que_head = ( recv_pkt_que_head + 1 ) AND 07H:
recv_pkt_que_in = recv_data_pkt_buf( recv_pkt_que_head );

IF (( recv_pkt_que_head + 1 ) AND 07H) = recv_pkt_que_tail
THEN recv_pkt_que_full = 1;
END;

END enqueue;

trans_ack = recv_sequence;
IF recv_pkt_que_full
THEN DO;
trans_ack = trans_ack OR ack_flow_field;
recv_flow_control = 1;
END;
ok = 1;
watch_dog_timer = watch_dog_timer_restart;
recv_sequence = ( recv_sequence + 1 ) AND 03H;
END;
ELSE IF recv_data_pkt_buf( recv_pkt_que_head ).control = (recv_sequence-1) AN
D 03H
THEN DO;
trans_ack = ( recv_sequence - 1 ) AND 03H;
IF recv_pkt_que_full
THEN DO;
trans_ack = trans_ack OR ack_flow_field;
recv_flow_control = 1;
END;
ok = 1;
watch_dog_timer = watch_dog_timer_restart;
END;
ELSE IF trans_ack = control_open_session
THEN DO;
IF NOT session_active
THEN DO;
IF recv_data_pkt_buf( recv_pkt_que_head ).destination_address = local_addr
THEN DO:
remote_addr = stat_session_via_remote;
CALL session_open;
trans_ack = ack_ok;
ok = 1;
watch_dog_timer = watch_dog_timer_restart;
END;
ELSE DO;
EL1SE IF trans_ack = control_close_session
THEN DO:
IF session_active
THEN IF recv_data_pkt_buf( recv_pkt_que_head ).destination_address = local_addr
THEN DO;
trans_ack = ack_ack;
ok = 1;
watch_dog_timer = watch_dog_timer_restart;
END:
ELSE IF trans_ack = control_flow_off
THEN DO;
ELSE IF session_active
THEN IF recv_data_pkt_buf( recv_pkt_que_head ).destination_address = local_addr
THEN DO;
trans_flow_control = 0;
trans_ack = ack_ok;
ok = 1;
watch_dog_timer = watch_dog_timer_restart;
/* Restart transmitter. */
IF packet_trans_wait
THEN EXIT = 1;
END;

ELSE IF trans_ack = control_watch_dog
THEN DO;
IF session_active
THEN IF recv_data_pkt_buf( recv_pkt_queue ).destination_address = local_addr
THEN DO;
ok = 1;
watch_dog_timer = watch_dog_timer_restart;
END;
ELSE IF tran_ck = control_watch_dog
THEN DO;
ELSEIF recv_data_pkt_buf( recv_pkt_queue ).source_address = remote_addr
THEN DO;
ELSE IF trans_ack = ack_ok;
ok = 1;
watch_dog_timer = watch_dog_timer_restart;
ELSE IF payload valid, ignore the packet
ELSE IF recv_pkt_queue_out PROCEDURE PUBLIC IS;
/*---------------------------------------------------------------------------*/
/* Procedure to dequeue a data block from the net receive queue. On return recv_pkt_queue_out points to the data block just dequeued. */
/*---------------------------------------------------------------------------*/
IF recv_pkt_queue_empty
/* If the queue is empty when called, then do nothing; return null pointer. */
THEN recv_pkt_queue_out = 0;
ELSE DO;
DISABLE;
recv_pkt_queue_empty = 0;
recv_pkt_queue_tail = (recv_pkt_queue_tail + 1) AND 07H;
recv_pkt_queue_out = recv_data_pkt_buf( recv_pkt_queue_tail ).data_length;
IF ((recv_pkt_queue_tail + 1) AND 07H) = recv_pkt_queue_head
THEN recv_pkt_queue_empty = 1;
ELSE DO;
DISABLE;
recv_pkt_queue_out = recv_data_pkt_buf( recv_pkt_queue_tail ).data_length;
IF recv_flow_control
THEN IF ((recv_pkt_queue_head - recv_pkt_queue_tail) AND 07H)
THEN DO; /* Build and send a Flow_Off packet */
control_flow_off_pkt_buf.destination_address = remote_addr;
control_flow_off_pkt_buf.source_address = local_addr;
END; /* If the queue is not empty, then dequeue a data block. */
ELSE DO;
DISABLE;
recv_pkt_queue_empty = 0;
recv_pkt_queue_tail = (recv_pkt_queue_tail + 1) AND 07H;
recv_pkt_queue_out = recv_data_pkt_buf( recv_pkt_queue_tail ).data_length;
IF ((recv_pkt_queue_tail + 1) AND 07H) = recv_pkt_queue_head
THEN recv_pkt_queue_empty = 1;
PL/M-61 COMPILER  WEILAT Network Layer Module

control_flow_off_pkt_buf.control = control_flow_off;
control_flow_off_pkt_buf.data_length = 0;
request_flow_off = 1;
If packet_trans_wait
  THEN Ex1 = 1;
END;

END enable;
END;
END recv_pkt_que_deque;
END netlay;

MODULE INFORMATION:

<table>
<thead>
<tr>
<th>Code Size</th>
<th>Constant Size</th>
<th>Direct Variable Size</th>
<th>Indirect Variable Size</th>
<th>Bit Size</th>
<th>Bit-Addressable Size</th>
<th>Auxiliary Variable Size</th>
<th>Maximum Stack Size</th>
<th>Register-Bank(s) Used</th>
<th>022 Lines Read</th>
</tr>
</thead>
<tbody>
<tr>
<td>00FFH</td>
<td>0000H</td>
<td>12H+00H</td>
<td>00H+00H</td>
<td>01H+03H</td>
<td>00H+00H</td>
<td>0028H</td>
<td>0010H</td>
<td>0 1</td>
<td></td>
</tr>
</tbody>
</table>

END OF PL/M-61 COMPILATION
Appendix H
Data Link Layer Module

PLM-61 COMPILER    DKLAY Data Link Layer Module

ISIS-II PLM-61 V1.1
COMPILER INVOKED BY: PLM61 :PS:DNLAY.JNK

$PAGEWIDTH( 95 )
$TITLE('DLKALY Data Link Layer Module ')
$REGISTERBANK( 1 )

1   dklay: DO:
     /===============================================================================*/
     /* dklay                                 */
     /* = Dave Wickliff                        */
     /* = Last Modified: 11/27/86               */
     /* =                                             */
     $NOLIST

     / Location of the data_length field in a packet */
8 1   DECLARE field_data_length  LITERALLY '06H';
8 1   DECLARE bad_ack             LITERALLY '0F8'; /* An invalid ack */

8 1   DECLARE mbr_status_led      BIT AT( 08H ) REGISTER;
9 1   DECLARE transmitter_control BIT AT( 084H ) REGISTER;

9 1   DECLARE max_backoff_count   LITERALLY '07H';
10 1  DECLARE max_defer_count    LITERALLY '07H';

/* TH=255, SMOD=1 => 82.50MHz bit rate on the net, for 12MHz clock. */
11 1  DECLARE net_rate_timer_restart LITERALLY '255';

152
DECLARE session_active BIT EXTERNAL;

/* Queue procedures and variables */
trans_pkt_que_deque: PROCEDURE BIT EXTERNAL; END;
DECLARE trans_pkt_que_out WORD EXTERNAL;
DECLARE recv_ack BYTE EXTERNAL;
recv_pkt_que_enqueue: PROCEDURE BIT EXTERNAL; END;
DECLARE recv_pkt_que_in WORD EXTERNAL;
DECLARE trans_ack BYTE EXTERNAL;

/* Externals from TIMSRV */
timer variables and restart values
DECLARE backoff_timer BYTE EXTERNAL;
DECLARE backoff_timer_restart LITERALLY '001H';
DECLARE net_jammed_timer BYTE EXTERNAL;
DECLARE net_jammed_timer_restart LITERALLY '002H';
DECLARE trans_defer_timer BYTE EXTERNAL;
DECLARE trans_defer_timer_min LITERALLY '07H';
DECLARE ack_timer BYTE EXTERNAL;
DECLARE ack_timer_restart LITERALLY '0FH';
DECLARE carrier_timer BYTE EXTERNAL;
DECLARE carrier_timer_restart LITERALLY '10H';
DECLARE blink_timer BYTE EXTERNAL;
DECLARE blink_timer_restart LITERALLY '80H';

$INCLUDE(ef8:errmsg.dcl )
/* Error/Status messages for HEILAY */
DECLARE error_retry LITERALLY '01H';
DECLARE error_retry_msg LITERALLY 
   "Retry exceeded, No response from remote.";
DECLARE error_watch_dog LITERALLY '02H';
DECLARE error_watch_dog_msg LITERALLY 
   "Watch dog time out, No response from remote.";
DECLARE state_session_via_remote LITERALLY '10H';
DECLARE state_session_via_remote_msg LITERALLY 
   "Session established via remote.";
DECLARE state_session_via_local LITERALLY '11H';
DECLARE state_session_via_local_msg LITERALLY 
   "Session established via local.";
DECLARE stat_remote_busy LITERALLY '12H';
DECLARE stat_remote_busy_msg LITERALLY 
   "Remote is busy.";
DECLARE stat_remote_not_available LITERALLY '13H';
DECLARE stat_remote_not_available_msg LITERALLY 
   "Remote is not available.";
DECLARE state_close_via_local LITERALLY '14H';
DECLARE stat_close_via_local_mag LITERALLY "Session closed via local.";
DECLARE stat_close_via_remote LITERALLY "Session closed via remote.";

DECLARE error_backoff_mag LITERALLY "Backoff retry exceeded.";
DECLARE error_jammed LITERALLY "Network jammed.";
DECLARE error_defer_mag LITERALLY "Defeat timeout.";

DECL.lRE pac.k•t_r cvwait BIT;
DECLARE packet_retcv_char BIT;
DECLARE packet_retcv_ack BIT;

DECLARE packet_trans_wait BIT PUBLIC;
DECLARE packet_trans_pending BIT;
DECLARE packet_trans_defer BIT;
DECLARE packet_trans_char BIT;
DECLARE packet_trans_ack BIT;

DECLARE data_link_status BYTE PUBLIC;
DECLARE random BYTE PUBLIC;

DECLARE c1 count BYTE;
DECLARE trans_terminal_count BYTE;
DECLARE recv_terminal_count BYTE;
DECLARE check_sum BYTE;
DECLARE tchar BYTE;
DECLARE rchar BYTE;
DECLARE last_tchar BYTE;
DECLARE backoff_count BYTE PUBLIC;
DECLARE defer_count BYTE PUBLIC;
DECLARE packet_ptr WORD;
DECLARE junk_bit BIT;

data_initialize: PROCEDURE PUBLIC USING O;
DECLARE packet_ptr WORD;
DECLARE packet BASED packet_ptr BYTE AUXILIARY;

packet_ptr = trans_pkt_que_out;

packet_ptr = packet_ptr + field_data_length;
trans_terminal_count = packet + field_data_length +1;

defer_count = 0;
backoff_count = 0;

IF carrier_timer <> 0 THEN DO;
packet_trans_pending = 1;

jammed_timer = jammed_timer_restart;
END;
END initiate_packet_trans;

PROCEDURE indirectly_callable public;

/* Procedure to service a carrier-lost "interrupt". Called by the
  timer interrupt service routine.
*/

/* Loose carrier when waiting to transmit a packet, then begin trans defer */
IF packet_trans_pending THEN DO;
  packet_trans_defer = 1;
  /* Defer a random time */
  trans_defer_timer = ( OSH AND random ) + trans_defer_timer_min;
END; /* then */

/* Loose carrier when receiving a packet, then abort and ignore packet */
IF packet_recv_char THEN DO;
  packet_recv_char = 0;
  packet_recv_wait = 1;
END;

PROCEDURE indirectly_callable public;

/* Procedure to service a backoff-timeout "interrupt". Called by the
  timer interrupt service routine.
*/

/* carrier_lost_serv: PROCEDURE INDIRECTLY_CALLABLE PUBLIC;
="/---------------------------------------------------------------------------"
IF carrier_timer <> 0 THEN DO;
  packet_trans_defer = 1;
  /* Defer a random time */
  trans_defer_timer = ( OSH AND random ) + trans_defer_timer_min;
END;
ELSE DO;
  packet_trans_defer = 1;
  /* Defer a random time */
  trans_defer_timer = ( OSH AND random ) + trans_defer_timer_min;
END;

PROCEDURE indirectly_callable public;

END backoff_serv;

156
defer_serv: PROCEDURE INDIRECTLY_CALLABLE PUBLIC;
/*-----------------------------------------------*/
/* defer_serv */
/* Procedure to service a transmit-defer-timeout "interrupt". Called by */
/* the timer interrupt service routine. */
/*-----------------------------------------------*/

net_jammed_timer = 0;
packet_trans_defer = 0;

IF carrier_timer <> 0
/* If net activity, then defer again until no activity */
THEN IF defer_count < max_defer_count
    THEN DO:
        defer_count = defer_count + 1;
packet_trans_pending = 1;
net_jammed_timer = net_jammed_timer_restart;
END;
ELSE DO:
    /* If too many defer trials, then give up and abort packet trans. */
    ELSE DO:
data_link_status = error_defer;
packet_trans_wait = 1;
junk_bit = trans_pkt_que_deque; /* Inform Network Layer. */
END;

/* If no net activity, then begin packet transmission */
ELSE DO:
packet_trans_char = 1;
char_count = 0;
check_sum = 0;
packet_ptr = trans_pkt_que_start;
usr_status_led = NOT session_active; /* Blink the status LED */
blink_timer = blink_timer_restart;
transmitter_control = 0; /* Line Driver ON */
TI = 1;
END;

END defer_serv;

ack_serv: PROCEDURE INDIRECTLY_CALLABLE PUBLIC;
/*-----------------------------------------------*/
/* ack_serv */
/* Procedure to service a waiting-for-ack timeout "interrupt". Called by */
/* the timer interrupt service routine. */
/*-----------------------------------------------*/

/* No ack received, pass a bad ack back to the Network layer */
recv_ack = bad_ack;
/* and get the next packet to transmit */
packet_trans_ack = 0;
IF trans_pkt_que_deque
    THEN CALL initialize_packet_trans; /* If there is a pkt, start sending */
ELSE packet_trans_wait = 1; /* otherwise stop the transmitter. */
PL/M-51 COMPILER   DLXLAY Data Link Layer Module

178 1 END task_serv;

177 2 jammed_serv: PROCEDURE INDIRECTLY_CALLABLE PUBLIC;
    /*-----------------------------------------------*/
    /* jammed_serv */
    /* Procedure to service a net-jammed-timeout "interrupt". Called by the */
    /* timer interrupt service routine. */
    /* */
    data_link_status = error_jammed;
179 2 packet_trans_pending = 0;
180 2 packet_trans_wait = 1;
181 2 junk_bit = trans_pkt_que_deque; /* Inform Network layer of the error */

182 1 END jammed_serv;

183 2 restart_serv: PROCEDURE INDIRECTLY_CALLABLE INTERRUPT 2;
    /*-----------------------------------------------*/
    /* restart_serv */
    /* Procedure to service interrupts from External Interrupt 1 of the 8751. */
    /* the Restart Transmitter Interrupt. Since the INT1 pin is tied low, this */
    /* interrupt can be generated via software by setting the interrupt enable */
    /* bit, Ei. */
    /* */
184 2 Ei = 0; /* Disable the interrupt */
185 2 IF packet_trans_wait /* Initiate a new packet trans */
    THEN IF data_link_status = 0
186 3 THEN IF trans_pkt_que_deque
187 3 THEN DO;
188 3 packet_trans_wait = 0;
189 3 CALL initiate_packet_trans;
190 3 END;
191 3 
192 1 END restart_serv;

193 2 serial_serv: PROCEDURE INDIRECTLY_CALLABLE INTERRUPT 4;
    /*-----------------------------------------------*/
    /* serial_serv */
    /* Procedure to service interrupts from the Network Port (the 8751 */
    /* internal serial port). */
    /* */
    DECLARE packet_based packet_ptr BYTE AUXILIARY;
    /*-----------------------------------------------*/
    /* Main code of serial_serv. */
    /* */
194 2 IF RI /* If a receive interrupt */
    THEN DO;
195 3 RI = 0;
196 2 IF carrier_timer = 0 /* Update carrier state. */
    THEN
PL/M-61 COMPILER  DIXILAY Data Link Layer Module

189 4        carrier_detect_service: DO;
             /*------------------------------------------*/
             /* carrier_detect_service */
             /* Procedure to service a carrier-detect interrupt. Called by the */
             /* serial interrupt service routine. */
             /*
             */
             /* If not transmitting a packet */
             IF NOT ( packet_trans_char or packet_trans_ack )
             /* Start receiving the packet */
             THEN DO;
     202 5           packet_ptr = recv_pkt_queue_in;
     203 5           packetrecv_wait = 0;
     204 5           char_count = 0;
     205 5           check_sum = 0;
     206 5           recv_terminal_count = 7fH;
     207 5           packetrecv_char = 1;
     208 5           END; /* then */
     209 3           END carrier_detect_service;
     210 3           carrier_timer = carrier_timer_restart;
     211 3           IF packet_recv_char
             /* Receive a character for a packet. */
             THEN
             /*
             */
     212 4           pkt_recv_service: DO;
             /*------------------------------------------*/
             /* pkt_recv_service */
             /*
             */
     213 4           rchar = SBUF;
             /* Read the received character */
             /* The following block of code could be included to reduce the receiver's */
             /* overhead by ignoring any packets not addressed to the node. */
             IF char_count = 0
             THEN IF rchar <> HIGH( node_addr )
             THEN DO;
             packet_recv_char = 0;
             packet_recv_wait = 1;
             END;
             IF char_count = 1
             THEN IF rchar <> LOW( node_addr )
             THEN DO;
             packet_recv_char = 0;
             packet_recv_wait = 1;
             END;
             /*
             */
     214 4           IF char_count = field_data_length
             THEN recv_terminal_count = rchar + field_data_length + 1;
     215 4           IF char_count = recv_terminal_count
             THEN IF check_sum = rchar
THEN DO;
   /* If check sum ok, then prepare to acknowledge the packet */
   packet_recv_char = 0;
   packet_recv_ack = 1;
   char_count = 0;
   TI = 1;
END;
ELSE DO;
   /* Else if check sum is bad, then ignore packet */
   packet_recv_char = 0;
   packet_recv_wait = 1;
END;
ELSE DO:
   check_sum = check_sum + rchar;
   packet = rchar;
   packet_ptr = packet_ptr + 1;
   char_count = char_count + 1;
END;
END pkt_recv_serv;
ELSE IF packet_trans_ack
   /* Or receive an ACK. */
   THEN
ack_recv_serv: DO;
   /*-----------------------------------------------*/
   /* ack_recv_serv */
   /* Procedure to receive an acknowledgment for a transmitted packet. */
   /* */
ack_timer = 0;  /* Turn off timer */
recv_ack = SBUF;  /* Read the received character */
/* Process receives ACK and gets the next packet to transmit */
packet_trans_ack = 0;
IF trans_pkt_que_deque THEN CALL initiate_packet_trans; /* If there is a pkt, start sending */
ELSE packet_trans_wait = 1;  /* otherwise stop the transmitter */
END ack_recv_serv;
END;
IF TI  /* If a transmit interrupt */
THEN DO;
TI = 0;
IF packet_trans_char  /* Send the next char of the packet */
THEN
pkt_trans_serv: DO;
/*-----------------------------------------------*/
/* pkt_trans_serv */
/* */
/* If the last char of the packet has not been sent, then send the next char. */

260 4 IF char_count <> trans_terminal_count + 1
261 5 THEN DO;
262 6 tchar = packet;
263 5 SBUF = tchar;
264 5 check_sum = check_sum + tchar;
265 5 END;

/* See if the char sent last time was received. If not then error. */
266 4 IF char_count <> 0
267 5 THEN DO;
268 6 rchar = SBUF;
269 5 RI = 0; /* Kill a possible recv interrupt */
270 5 IF rchar <> last_tchar
271 6 /* If error, then try backing off and restart transmission later. */
272 6 THEN IF backoff_count < max_backoff_count
273 7 THEN DO;
274 8 backoff_timer = backoff_timer_restart;
275 8 backoff_count = backoff_count + 1;
276 8 packet_trans_char = 0;
277 8 packet_trans_backoff = 1;
278 8 END;
279 8 ELSE DO;
280 9 data_link_status = error_backoff;
281 8 packet_trans_char = 0;
282 8 packet_trans_wait = 1;
283 8 junk_bit = trans_pkt_que_deque; /* Inform Network */
284 8 /* Layer of the error. */
285 8 END;

/* If too many backoff tries, then give up and abort packet trans. */
286 6 ELSE DO;
287 7 data_link_status = error_backoff;
288 6 packet_trans_char = 0;
289 6 packet_trans_wait = 1;
290 6 junk_bit = trans_pkt_que_deque; /* Inform Network */
291 6 /* Layer of the error. */
292 6 END;

293 4 IF packet_trans_char /* Do not execute this block if there was */
294 5 THEN DO; /* an error */
295 6 /* If the last data character was transmitted, then insert current */
296 6 /* check sum into the packet for next character transmission. */
297 7 IF char_count + i = trans_terminal_count
298 8 THEN DO;
299 9 packet_ptr = packet_ptr + 1;
300 9 packet = check_sum;
301 8 packet_ptr = packet_ptr - 1;
302 8 END;
303 8 /* If the packet has been successfully transmitted, then wait for the */
304 9 /* next packet. */
305 8 IF char_count = trans_terminal_count + 1
306 9 THEN DO;
THEN DO;
    transmitter_control = 1;  /* Line Driver OFF */
    ack_timer = ack_timer_restart;
    packet_trans_char = 0;
    packet_trans_ack = 1;
END;  /* END THEN */

/* If not done transmitting current packet, then continue transmitting. */
ELSE DO;
    char_count = char_count + 1;
    packet_ptr = packet_ptr + 1;
    last_tchar = tchars;
END;

END pkt_trans_serv;

ELSE IF packet_recv_ack /* Or send an ACK. */
THEN.

ack_trans_serv: DO;
/*--------------------------------------------------------------------------*/
/* ack_trans_serv */
/* Procedure to send an acknowledgment for a received packet. */
/* */
ack_trans_serv: DO;

IF char_count = 0
THEN DO;
    /* Enqueue the packet and compute acknowledgment */
    IF recv_pkt_queue
THEN DO;  /* Send the acknowledgment if there is one */
        transmitter_control = 0;  /* Line Driver ON */
        user_status_led = NOT session_active;  /* Blink the status LED */
        blink_timer = blink_timer_restart;
        SBUF = trans_ack;
        char_count = 1;
        END;
    ELSE DO;  /* Otherwise, send nothing */
        packet_recv_ack = 0;
        packet_recv_wait = 1;
        END;
ELSE DO;
    rchar = SBUF;
    transmitter_control = 1;  /* Line Driver OFF */
    packet_recv_ack = 0;
    packet_recv_wait = 1;
    END;
END ack_trans_serv;
PL/M-61 COMPILE

DILKAY Data Link Layer Module

321. 3 END;
322 1 END serial_serv;
323 1 END diplay;

WARNINGS:
4 IS THE HIGHEST USED INTERRUPT

MOBILE INFORMATION:

<table>
<thead>
<tr>
<th>(STATIC+OVERLAYABLE)</th>
<th>CODE SIZE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>= 0260H</td>
</tr>
<tr>
<td></td>
<td>= 592D</td>
</tr>
<tr>
<td>CONSTANT SIZE</td>
<td>= 0000H</td>
</tr>
<tr>
<td>DIRECT VARIABLE SIZE</td>
<td>= ODH+02H</td>
</tr>
<tr>
<td>INDIRECT VARIABLE SIZE</td>
<td>= 00H+0DH</td>
</tr>
<tr>
<td>BIT SIZE</td>
<td>= 0AH+00H</td>
</tr>
<tr>
<td>BIT-ADDRESSABLE SIZE</td>
<td>= 00H+00H</td>
</tr>
<tr>
<td>AUXILIARY VARIABLE SIZE</td>
<td>= 000H</td>
</tr>
<tr>
<td>MAXIMUM STACK SIZE</td>
<td>= 000FH</td>
</tr>
<tr>
<td>REGISTER-BANK(S) USED:</td>
<td>0 1</td>
</tr>
</tbody>
</table>

656 LINES READ
0 PROGRAM ERROR(S)
END OF PL/M-61 COMPILATION
Appendix I
Timer Server Module

PL/M-61 COMPILER TIMSRV - Timer Interrupt Server Module

ISIS-II PL/M-61 V1.1
COMPILER INVOKED BY: PLM61 :F8:TIMSRV.J8K

$PAGEWIDTH( 95 )
$TITLE('TIMSRV - Timer Interrupt Server Module ')
$REGESTERBANK( 4 )

1 1 timer: DO;

/*---------------------------------------------------------------------------*/
/* timer */

/*---------------------------------------------------------------------------*/
/* Externals from DKLAY */
5 2 tdefe_serv: PROCEDURE EXTERNAL; END;
7 2 backoff_serv: PROCEDURE EXTERNAL; END;
9 2 jammed_serv: PROCEDURE EXTERNAL; END;
11 2 tuck_serv: PROCEDURE EXTERNAL; END;
13 2 carrier_lost_serv: PROCEDURE EXTERNAL; END;

/*---------------------------------------------------------------------------*/
/* Externals from WILAY */
15 2 watch_dog_serv: PROCEDURE EXTERNAL; END;
17 1 DECLARE session_active BIT EXTERNAL;
DECLARE char_rate_timer_restart LITERALLY '64736';

/**
 * Timer variables
 */
DECLARE backoff_timer BYTE PUBLIC;
DECLARE net_jammed_timer BYTE PUBLIC;
DECLARE trans_defer_timer BYTE PUBLIC;
DECLARE ack_timer BYTE PUBLIC;
DECLARE carrier_timer BYTE PUBLIC;
DECLARE buffer_timer BYTE PUBLIC;
DECLARE buffer_timer_restart BYTE PUBLIC;
DECLARE carrier_timer BYTE PUBLIC;
DECLARE ear_time BYTE PUBLIC;
DECLARE buffer_timer_restart BYTE PUBLIC;
DECLARE ear_time BYTE PUBLIC;
DECLARE blink_timer BYTE PUBLIC;
DECLARE watch_dog_timer BYTE PUBLIC;

PROCEDURE PUBLIC USING O;

PROCEDURE PUBLIC USING O;

PROCEDURE PUBLIC USING O;

PROCEDURE PUBLIC USING O;

PROCEDURE INTERRUPT I INDIRECTLY-CALLABLE;

PROCEDURE INTERRUPT I INDIRECTLY-CALLABLE;

DECLARE long_delay BYTE;
PL/M-61 COMPILER   TIMSrv - Timer Interrupt Server Module

/*-----------------------------------------------*/
/* Main code of tim_serve */
/*
45 2  /* Reset Timer 0 */
47 2  TRO = 0;
48 2  TL0 = LOW( char_rate_timer_restart );
49 2  TH0 = HIGH( char_rate_timer_restart );
50 2  TRO = 1;

/* Service "short" timers */
52 2  DISABLE;

/* Carrier "lost" time out */
54 2  IF carrier_timer <> 0
55 3  THEN Do;
56 3  carrier_timer = carrier_timer - 1;
57 3  IF carrier_timer = 0
58 3  THEN CALL carrier_lost_serve; /* Service carrier lost */
59 3  END;

/* Transmit defer timer */
61 2  IF trans_defer_timer <> 0
62 3  THEN Do;
63 3  trans_defer_timer = trans_defer_timer - 1;
64 3  IF trans_defer_timer = 0
65 3  THEN CALL tdefer_serve;
66 3  END;

/* ACK timer */
68 2  IF ack_timer <> 0
69 3  THEN Do;
70 3  ack_timer = ack_timer - 1;
71 3  IF ack_timer = 0
72 3  THEN CALL tack_serve;
73 3  END;
74 2  ENABLE;

/* Buffer packing timer */
76 2  IF buffer_timer <> 0
77 3  THEN IF ( long_delay AND OFH ) = 0 /* Divide by 16 */
78 3  buffer_timer = buffer_timer - 1;

/* User Status LED blink timer */
79 2  IF blink_timer <> 0
80 3  THEN Do;
81 3  blink_timer = blink_timer - 1;
82 3  IF blink_timer = 0
83 3  THEN user_status_led = session_active;
84 3  END;

/* Service "long" timers */
86 2  long_delay = long_delay-1;
87 2  IF long_delay = 0
88 2  /*
89 2  */
90 2  */
PL/I-61 COMPILER  TIMSERV - Timer Interrupt Server Module

THEN DO;
   82 3  DISABLE;

   /* Backoff timer */
   83 3  IF backoff_timer <> 0
       THEN DO;
       85 4  backoff_timer = backoff_timer - 1;
       86 4  IF backoff_timer = 0
       THEN CALL backoff_serv;
   88 4  END;

   /* Network jammed timer */
   89 3  IF net_jammed_timer <> 0
       THEN DO;
       91 4  net_jammed_timer = net_jammed_timer - 1;
       92 4  IF net_jammed_timer = 0
       THEN CALL jammed_serv;
       94 4  END;

   /* Watch Dog timer */
   95 3  IF watch_dog_timer <> 0
       THEN DO;
       97 4  watch_dog_timer = watch_dog_timer - 1;
       98 4  IF watch_dog_timer = 0
       THEN CALL watch_dog_serv;
   100 4  END;

   101 3  ENABLE;
   102 3  END;
   103 1  END tim_serv;
   104 1  END timsrv;

WARNING:
  1 IS THE HIGHEST USED INTERRUPT

MODULE INFORMATION: (STATIC+OVERLAYABLE)
CODE SIZE          = 00ACH  172D
CONSTANT SIZE      = 0000H  0D
DIRECT VARIABLE SIZE = 09B+01H  9D+ 1D
INDIRECT VARIABLE SIZE = 00H+00H  0D+ 0D
BIT SIZE           = 00H+00H  0D+ 0D
BIT-ADDRESSABLE SIZE = 00H+00H  0D+ 0D
AUXILIARY VARIABLE SIZE = 0000H  0D
MAXIMUM STACK SIZE = 0000FR  150
REGISTER-BANK(S) USED:  0 1
269 LINES READ
0 PROGRAM ERROR(S)

END OF PL/I-61 COMPILATION
References Cited


Unless otherwise specified, capacitance values are in microfarads and resistance values are in ohms (1/4W, 5%).

Power connections that are not shown on schematics:

<table>
<thead>
<tr>
<th>IC9</th>
<th>pin 7</th>
<th>pin 14</th>
<th>+12v</th>
<th>-12v</th>
</tr>
</thead>
<tbody>
<tr>
<td>IC10</td>
<td>pin 7</td>
<td>pin 14</td>
<td>+12v</td>
<td>-12v</td>
</tr>
<tr>
<td>IC11</td>
<td>pin 7</td>
<td>pin 14</td>
<td>+12v</td>
<td>-12v</td>
</tr>
</tbody>
</table>

Jumper options:

<table>
<thead>
<tr>
<th>Jumper Connections</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>E4 to E5</td>
<td>8751 processor</td>
</tr>
<tr>
<td>E5 to E6</td>
<td>8032 processor</td>
</tr>
<tr>
<td>E1 to E2</td>
<td>HM6264 CMOS RAM</td>
</tr>
<tr>
<td>E2 to E3</td>
<td>HM6116 CMOS RAM</td>
</tr>
<tr>
<td>E14 to E15</td>
<td>FG connected to SG</td>
</tr>
<tr>
<td>E12 to E13</td>
<td>On-card transceiver with IC13 and IC14 installed and IC12 removed</td>
</tr>
<tr>
<td>E10 to E11</td>
<td>E13 and IC14 installed</td>
</tr>
<tr>
<td>E7 to E8</td>
<td>Off-card transceiver with IC13 and IC14 installed and IC12 removed</td>
</tr>
<tr>
<td>E8 to E9</td>
<td>Off-card transceiver with IC13 and IC14 installed and IC12 removed</td>
</tr>
</tbody>
</table>

UNIVERSITY OF ILLINOIS
ADVANCED DIGITAL SYSTEMS LAB
LAN NODE CARD
EDGE CONNECTOR