A DEVELOPMENT SYSTEM FOR THE BUS MONITOR UNIT
FOR THE DATAC DIGITAL DATA BUS,

A Thesis Presented to
The Faculty of the College of Engineering and Technology
Ohio University

In Partial Fulfillment
of the
Requirements for the Degree
Master of Science

by
Stanley M. Novacki, III
November, 1987

OHIO UNIVERSITY
LIBRARY
Acknowledgements

To those individuals who have contributed to the success of this project but whose names do not appear below, please accept my apologies. It is not your contribution that is insignificant, but rather my ability to recall the numerous persons who have generously given of themselves. I may not be able to explicitly acknowledge your contribution, but please be assured that in one sense or another I have felt the effects of your help and thank you for your efforts on my behalf.

I would like to offer my thanks to the following persons for their contributions to this research effort: to Dr. Robert W. Lilley, for directing me to this program and providing an avenue of escape from 208V, 30; to Mr. Richard N. Turner, Avionics Research Engineer, for his clever utilities that greatly simplified testing the Bus Monitor Unit; and to Mr. James Ramsay and Mr. William Lynn of Kentron International (Hampton, Virginia) for spending their valuable time freely, providing information about the DATAC system.

Special thanks are due Ms. Mary Law, Avionics Research Engineer, for her unflagging encouragement; to Mr. Robert J. Thomas, Avionics Research Intern, for his enthusiastic assistance with the Bus Monitor hardware and software; to Mr. Mark Shafer of Bell Laboratories (Columbus, Ohio) for opening the sometimes ponderous doors to the C language; and to Mr. Stephen R. Yost of Rho, Incorporated and Mr. Robert E. Ray of Sikorsky Aircraft Corporation for the wealth of data they provided on the MIL-STD 1553B and ARINC 429 data busses.

I am indebted to: Mr. Douglas Althauser for his counsel during
the preparation of this work; Ms. Alicya Shade for her invaluable
direction and expert assistance during the production of this thesis;
and particularly to Mr. Brian D. Shapiro and Ms. Linda Yosic for
always being there when I was.

I especially would like to thank my parents to whom this thesis
is dedicated. Without their constant love and support it would not
have been possible.

This research effort and thesis was produced with support from the
National Aeronautics and Space Administration, research grant number NGR
36-009-017.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Acknowledgements</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>List of Figures</td>
<td>v</td>
</tr>
<tr>
<td>List of Tables</td>
<td>vii</td>
</tr>
<tr>
<td><strong>I. INTRODUCTION AND STATEMENT OF PROBLEM</strong></td>
<td>1</td>
</tr>
<tr>
<td><strong>II. REVIEW OF DATAC BUS OPERATION AND BUS MONITOR APPLICATION</strong></td>
<td>4</td>
</tr>
<tr>
<td>A. The DATAC Digital Data Bus</td>
<td>4</td>
</tr>
<tr>
<td>1. Background</td>
<td>4</td>
</tr>
<tr>
<td>2. DATAC Capabilities</td>
<td>20</td>
</tr>
<tr>
<td>3. An Overview of Two Popular Aircraft Data Busses</td>
<td>22</td>
</tr>
<tr>
<td>a. MIL STD 1553B</td>
<td>22</td>
</tr>
<tr>
<td>b. ARINC 429</td>
<td>27</td>
</tr>
<tr>
<td>c. A Comparison of DATAC, ARINC 429 and MIL STD 1553B</td>
<td>31</td>
</tr>
<tr>
<td>B. The DATAC Bus Monitor</td>
<td>36</td>
</tr>
<tr>
<td>1. Need for the Bus Monitor</td>
<td>38</td>
</tr>
<tr>
<td>2. Proposed Bus Monitor Functions</td>
<td>38</td>
</tr>
<tr>
<td>a. Data Stream Display</td>
<td>39</td>
</tr>
<tr>
<td>b. Data Stream Interpretation</td>
<td>39</td>
</tr>
<tr>
<td>c. Echo-check</td>
<td>40</td>
</tr>
<tr>
<td>d. Transmission Schedule Confirmation</td>
<td>41</td>
</tr>
<tr>
<td><strong>III. IMPLEMENTATION OF THE BUS MONITOR DEVELOPMENT SYSTEM</strong></td>
<td>43</td>
</tr>
<tr>
<td>A. DATAC Bus Terminal Data Stream</td>
<td>49</td>
</tr>
<tr>
<td>1. The Subsystem Interface Simulation</td>
<td>52</td>
</tr>
<tr>
<td>2. Subsystem Interface Driver</td>
<td>54</td>
</tr>
<tr>
<td>3. Bus Terminal Data Stream Simulation</td>
<td>59</td>
</tr>
<tr>
<td>B. Z8000/PC Processing Configuration</td>
<td>63</td>
</tr>
<tr>
<td>1. File Transfer Utilities</td>
<td>66</td>
</tr>
<tr>
<td>2. Basic Z8002 Software Features</td>
<td>72</td>
</tr>
<tr>
<td>3. IBM PC Data Handler Routines</td>
<td>74</td>
</tr>
<tr>
<td>4. BlueBoard Interface for ZBUS/S100 Signal Interchange</td>
<td>76</td>
</tr>
<tr>
<td>a. A8–A15 Address Line Extension</td>
<td>77</td>
</tr>
<tr>
<td>b. S100/ZBUS DMA Transfer</td>
<td>79</td>
</tr>
<tr>
<td>c. Interrupt Vector Capture and DMA Management</td>
<td>86</td>
</tr>
</tbody>
</table>
TABLE OF CONTENTS (continued)

C. Summary 92

IV. TEST PROCEDURE AND RESULTS 94

A. Bus Monitor Unit Testing and Data Collection Procedures 95
B. Test Results and Assessment of COMSAVE Data 98
C. Validity of Bus Terminal Interface Simulation 108
D. Applicability of Test Results to Alternate Modes of Operation 110

V. CONCLUSIONS 113

VI. RECOMMENDATIONS 115

A. Limitations of the S100-based Bus Monitor 115
B. Alternatives to Serial Data Transfer 116
C. Alternatives to the S100 System 118
D. Bus Terminal Interface Support 119

REFERENCES 127

APPENDICES 130

A. DATA TERMINAL SSI PINOUT
B. ROCKWELL R6522 VIA PRODUCT DESCRIPTION 133
C. DTERM3: DATA TERMINAL SSI SIMULATION 146
D. GENZHEX/BIN2THEX: Y ASSEMBLER TO TEKHEX FORMATTER 154
E. TEKHEX FILE FORMAT AND TRANSFER PROTOCOL 159
F. Z8KLOADER: BUS MONITOR TEKHEX DOWNLOADER 162
G. SENDING: TEKHEX HOST DOWNLOADER PROGRAM 168
H. MPU8000 II MONITOR COMMAND SUMMARY 173
I. GENZHEX/MAKEZHEX: Y ASSEMBLER TO .ZHX FORMATTER 175
J. PIN DESCRIPTION OF IEEE-696/S100 BUS 179
K. PIN DESCRIPTION OF ZILOG Z-BUS 186
L. TXMSG: BUS MONITOR MESSAGE RELAY PROGRAM 190
M. COMSAVE: MESSAGE RECEIVER/STORAGE PROGRAM 195
N. DSCAN: MESSAGE TABULATOR/PREFORMATTER PROGRAM 207
O. MPU8000 II SCHEMATIC 213
P. TRUMP CARD SCHEMATIC 216
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Basic Layout of the DATAC Data Bus</td>
<td>7</td>
</tr>
<tr>
<td>2.2</td>
<td>Simplified Duplex Utility System Using DATAC Data Bus</td>
<td>8</td>
</tr>
<tr>
<td>2.3</td>
<td>DATAC Terminal Simplified Block Diagram</td>
<td>10</td>
</tr>
<tr>
<td>2.4</td>
<td>DATAC Operation - Mode A</td>
<td>14</td>
</tr>
<tr>
<td>2.5</td>
<td>DATAC Operation - Mode B</td>
<td>16</td>
</tr>
<tr>
<td>2.6</td>
<td>DATAC Operation - Mode C (SCAT-C Type)</td>
<td>19</td>
</tr>
<tr>
<td>2.7</td>
<td>Typical MIL STD 1553B Installation</td>
<td>23</td>
</tr>
<tr>
<td>2.8</td>
<td>MIL STD 1553B Command/Data/Status Word Formats</td>
<td>24</td>
</tr>
<tr>
<td>2.9</td>
<td>Manchester II Data Encoding</td>
<td>26</td>
</tr>
<tr>
<td>2.10</td>
<td>General Word Formats and Encoding Examples for ARINC 429</td>
<td>30</td>
</tr>
<tr>
<td>2.11</td>
<td>Voltage Levels for ARINC 429 Bit Encoding</td>
<td>32</td>
</tr>
<tr>
<td>2.12</td>
<td>Data Bit Encoding for ARINC 429 Transmissions Using Bipolar Three-Level RZ Coding</td>
<td>33</td>
</tr>
<tr>
<td>3.1</td>
<td>Block Diagram of Shared Interface RAM (SIR)</td>
<td>45</td>
</tr>
<tr>
<td>3.2</td>
<td>DATAC Terminal Bus Encoder/Decoder Block Diagram</td>
<td>48</td>
</tr>
<tr>
<td>3.3</td>
<td>M68000 CPU Programmer's Model</td>
<td>51</td>
</tr>
<tr>
<td>3.4</td>
<td>DATAC Terminal Simulator Implementation</td>
<td>53</td>
</tr>
<tr>
<td>3.5</td>
<td>Byte/Word Addressing in the M68000</td>
<td>55</td>
</tr>
<tr>
<td>3.6</td>
<td>Subsystem Interface for DATAC Terminal Simulator Address/Data/Interrupt Line Generation</td>
<td>56</td>
</tr>
<tr>
<td>3.7</td>
<td>Subsystem Interface Simulation Control Line Generation</td>
<td>58</td>
</tr>
</tbody>
</table>
LIST OF FIGURES (continued)

| Figure 3.8. | Bus Monitor Software Test System | Page |
| Figure 3.9. | Z8002 CPU Programmer's Model | 64 |
| Figure 3.10. | Z8001 CPU Programmer's Model | 65 |
| Figure 3.11. | Y Assembler Binary File Format | 67 |
| Figure 3.12. | "ZHX" File Format for Program Downloading to Bus Monitor | 68 |
| Figure 3.13. | Basic Bus Monitor Message Handling Routine | 71 |
| Figure 3.14. | S100/IEEE-696 Address/Data/Control Line Interface | 75 |
| Figure 3.15. | Z-BUS Bus Transfer Timing | 78 |
| Figure 3.16. | S100/IEEE-696 Bus Transfer Timing | 81 |
| Figure 3.17. | BlueBoard DMA Management and ID Tag Capture | 82 |
| Figure 4.1. | Percentage of Messages Processed By Bus Monitor as a Function of Message Word Count | 88 |
| Figure 4.2. | Average Frame Adjacency of Bus Messages as a Function of Message Word Count | 101 |
| Figure 4.3. | Average Data Bandwidth of Bus Monitor as a Function of Message Word Count | 102 |
| Figure 6.1. | CAM Dual-Port Memory Configuration | 103 |
| Figure 6.2. | CAM End-of-Message Signaling to Trump Card II | 121 |
| Figure 6.3. | Proposed Bus Monitor II Installation in PC Chassis | 123 |

vi
## LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 2.1</td>
<td>Summary of DATAC System Capabilities</td>
<td>21</td>
</tr>
<tr>
<td>Table 2.2</td>
<td>MIL STD 1553B Bus Controller Commands</td>
<td>28</td>
</tr>
<tr>
<td>Table 2.3</td>
<td>Summary of MIL STD 1553B Features</td>
<td>29</td>
</tr>
<tr>
<td>Table 2.4</td>
<td>ARINC 429 Mark 33 DITS Summary</td>
<td>34</td>
</tr>
<tr>
<td>Table 2.5</td>
<td>Summary of DATAC, ARINC, And MIL STD 1553B Features</td>
<td>37</td>
</tr>
<tr>
<td>Table 4.1</td>
<td>Hexadecimal Data Collected by COMSAVE Program from S100 System</td>
<td>97</td>
</tr>
<tr>
<td>Table 4.2</td>
<td>Sample Data Output by DSCAN Program</td>
<td>99</td>
</tr>
<tr>
<td>Table 4.3</td>
<td>Summary of Data Collected by COMSAVE/TxMSG Programs</td>
<td>104</td>
</tr>
<tr>
<td>Table 6.1</td>
<td>Comparison of Features: Z8002-Based Bus Monitor and Z8001-Based Bus Monitor II</td>
<td>126</td>
</tr>
</tbody>
</table>
I. INTRODUCTION AND STATEMENT OF PROBLEM

The Digital Autonomous Terminal Access Communications (DATAC) data bus is a data communications system designed by the Boeing Company for use aboard its future generations of passenger aircraft [1]. DATAC is intended to interconnect any systems which exchange data among aircraft systems. It is currently implemented in the Advanced Transportation Operations aircraft (ATOPS) under development at the NASA Langley Research Center at Hampton, Virginia. The ATOPS is a Boeing B737 fitted with a complete "fly-by-wire" system that uses DATAC as the communications medium for the various aircraft controls and instrumentation. This network uses a Carrier Sense Media Access with Collision Detection access protocol. A feature of this broadcast protocol is that access to the bus is granted autonomously. Various data are contained in packets called messages. Message routing is implemented by using an identification tag, called a label, prefixed to each message. The label indirectly identifies the source of the message. All aircraft systems using the bus receive the label and, based on its value, may then receive the message and process the data it contains.

Each bus terminal acts independently of the others on the bus when it transmits a message. A message sent out on the DATAC bus is not destined for a particular system. Rather, messages are broadcast by one system and received by other systems which are programmed to receive them. The transmitting system does not interact with other systems to arbitrate their communication duties. This requires that each bus system monitor itself and use the bus in an orderly fashion.
Because each system accesses the bus without arbitrating with another system specifically for the purpose of bus access, it is not possible to interrogate explicitly a specific bus system, whether to obtain data, determine the quality of its operation, or even confirm that the system in question is operating properly.

The purpose of the DATAC Bus Monitor Unit is two-fold. First it provides a means to observe bus transactions, and secondly it allows user-access to the other bus systems for purposes of determining the operating status of the bus. The Bus Monitor Unit must be able to receive every message which appears on the bus. It must be able to communicate with other bus systems to determine their operating status.

This paper documents the design, construction and testing of a microcomputer system which provides a test bed for developing hardware and software to implement a Bus Monitor Unit. An S100/IEEE-696 microcomputer system consisting of an Ithaca Intersystems MPU8000 II Z8002-based CPU card, 6S1O six-channel serial communications card, and a CompuPro RAM-16 64Kb memory card forms the basic Bus Monitor Unit main processor which receives data from the bus. An IBM Personal Computer (PC) Model 5150 serves as the console for the S100 system, providing command input, message display, and data reduction and storage capabilities. Because of the general unavailability of active DATAC systems to provide data, data flow from the DATAC bus to the Bus Monitor Unit is simulated using a Motorola MEX68KECB Educational Computer Kit, which is a 68000L4-based Single Board Computer (SBC). Tandem parallel ports and small-scale integration
logic devices on the SBC are used to provide the various data and control lines used by the Z8002 system's bus interface unit.

Tests performed with this system responding to a simulated DATAC data stream show that the S100/PC configuration is capable of providing limited Bus Monitor functions. Due to hardware constraints, this system can only process a fraction of the total data distributed by the bus. However, this system is useful in an environment where real-time processing of large volumes of data or high-speed throughput is not a consideration. Analysis of the test results identifies areas for improvement which, when addressed, will implement a Bus Monitor system capable of managing normal DATAC bus traffic as well as providing enhanced features such as independent reconstitution of bus data.
II. REVIEW OF DATA BUS OPERATION AND BUS MONITOR APPLICATION

A. The DATA BUS Digital Data Bus

1. Background. Rising operating and manufacturing costs have taken a heavy toll on commercial airframe manufacturers in the past decade. In order to counter these rising costs, aircraft manufacturers have responded by employing new technologies in aircraft design [2]. The use of composite materials, for example, results in substantial weight savings over comparable metal parts. Novel wing designs also promise improved performance over their conventional counterparts [3].

Another way of reducing operating costs is simply to reduce the number of non-revenue people aboard the aircraft [4]. However, the elimination of a co-pilot or flight engineer from the crew of a new-generation aircraft is a serious issue, considering the increasing complexity of aircraft operation in crowded metropolitan areas [5]. To maintain flyability without compromising safety, an obvious choice is to incorporate some type of computer system to assist in the operation of the aircraft. The small size, low cost and high performance of today's microcomputer systems make them naturally suited to airborne applications.

While the desirability of adding these systems may be readily apparent, the weight associated with them is somewhat more subtle. Not only is there the mass of the components themselves, but usually these computers must communicate with one another as well as with existing aircraft systems. This requires some type of communications channel, typically implemented as bundles of point-to-point wires.
Adding these computer units and support hardware can add greatly to the weight of an aircraft. When the full impact of the addition is assessed, it may be that in order to maintain the intended operating profile, any weight savings and gains in operating economy achieved by reduction of the crew complement and using new materials and technologies in construction and operation is offset by the weight of the computers, wiring, and other systems needed to manage these new aircraft. Clearly, there is a need for an approach to new aircraft and aircraft system design to retain the weight savings available from using these new procedures while maintaining the flyability of the aircraft.

A data communications system such as DATAC is one solution to these problems. DATAC provides a means for implementing a global data base for use by various computerized subsystems aboard current and next-generation commercial aircraft. We shall define a bus subsystem as a device or system that is an operating component of the aircraft; a bus system extends to include the DATAC bus terminal which serves to interface the subsystem to the bus. The expression "global data bus" describes DATAC's ability to provide access to all aircraft parameters by any subsystem that uses the bus for communication. This global (widespread) availability of all aircraft flight data obviates the need for point-to-point wiring, replacing such a complex interconnection scheme with a simple party-line approach. An alternative perspective would be to view DATAC as a local-area network intended for use with aircraft systems.

A DATAC system consists of the bus medium, the bus couplers and
bus terminals which enable the various subsystems to communicate via the bus. Figure 2.1 shows a block diagram of a DATAC system. The bus may be either a twisted pair employing bipolar voltages or a current loop, or an optical fiber. The bus couplers are devices which take the bit-serial output of the DATAC terminals and convert it to the type of signal appropriate for the bus medium in use. The bus terminals described below act as interfaces between the bus and the subsystem that uses the bus for data transfers. A subsystem may be a computer, a sensor, a display, a set of switches or relays, or a set of controls.

Figure 2.2 shows a possible DATAC installation aboard a commercial airliner. In this illustration, two redundant buses provide communications between such systems as engine control, aircraft control surfaces, the cockpit controls and instrumentation, and other subsystems.

As mentioned earlier, each aircraft or laboratory subsystem which uses the DATAC bus is connected to it by way of a bus coupler/bus terminal pair. The terminal functions as a data format converter and protocol manager, providing services similar to those obtained from a Universal Asynchronous Receiver/Transmitter (UART) commonly used in asynchronous computer communications. The bus terminal provides not only the parallel-serial data interchange of the UART but controls the data flow to and from the bus and determines what data is received from the bus and relayed to the attached subsystem. The terminal is responsible for sending any data from its attached subsystem across the bus in a timely fashion.
Figure 2.1: Basic Layout of the DATAC Data Bus
Figure 2.2. Simplified Duplex Utility System Using DATAC Data Bus
coordinating itself with the other transmitting bus terminals. It also provides an identifying label to the data that the bus terminal's subsystem wishes to broadcast via the bus. The actual protocol for message transmission is described in following sections.

A block diagram of a bus terminal is shown in figure 2.3. The terminal consists of two separate sections, a receiver and a transmitter. Each section contains a bus coupler and a protocol manager. The protocol manager monitors bus transactions to ensure that the correct data format is used and that only one system uses the bus at a given time.

The receiver section contains a bus receiver which collects data from the bus coupler and presents it bit-serially to the decoder which produces a 16-bit parallel data word. Similarly, the transmitter section contains an encoder which takes a 16-bit parallel word, adds start, stop, and sync bits, translates it to the corresponding 20-bit quantity in the Manchester II format and passes it to the transmitter which sends the data out serially onto the bus. Using the Manchester II coding scheme, the 20 bit-times used to form a data word result in a datum duration time of 20 μs [6].

In the case of a computer subsystem (the Bus Monitor is an example of this), the interface between the bus terminal and the computer would typically consist of a two-port Random Access Memory (RAM) device shared between the computer and the bus terminal. If the terminal presents data from the bus, the computer is alerted and reads the data. If the computer is to send data across the bus, it places that information into the shared RAM. It is then the
Figure 2.3. DATAC Terminal Simplified Block Diagram
terminal's responsibility to access the data, format it and place it on the bus.

The DATAC bus terminals use a Carrier Sense Multiple Access with Collision Detection (CSMA/CD) access protocol [7], similar to that employed by the commercial Ethernet Local Area Network [8]. This means that the key to each terminal's use of the bus is its ability to determine message activity on the bus. If a data transmission from another terminal is detected, the bus is in use and the terminal wishing to transmit should not attempt to access the bus. Should two bus terminals attempt to access the bus at the same time, each unit is designed to recognize the corruption of its message by the other terminal and abort the attempt, hence the term "collision detection."

In addition to the CSMA/CD protocol, DATAC employs time-division multiplexing. Detection of "bus quiet time" as described above is the first step in gaining access to the bus. The second step involves determining which system will access the bus at this particular time. Each bus terminal is equipped with a counter which counts to a value unique among all the terminals on the bus. This unique value is called the Terminal Gap (TG) value. As soon as the terminals detect a period of "bus quiet time," each terminal begins to count to its unique TG value. The first counter to reach its terminal value assumes the right to access and the bus terminal begins to transmit. When this occurs, the other terminals detect bus activity and reset their gap timers, waiting for the next occurrence of "bus quiet" so that they will again begin counting to their unique
TG values and attempt to access the bus.

Using a single timer to manage access contention might seem to allow the first system to have timed-out during the previous quiet-time to be the next terminal to broadcast, time after time. A mechanism is provided to avoid this situation. After a terminal has completed a transmission, its own timer is inhibited from counting until at least one other timing qualifier, either another counter or an external event, has occurred. The number of qualifiers depends on the mode of operation. This ensures that no one device will monopolize the bus by first allowing all other terminals the opportunity to access the bus before the terminal with the shortest TG value may transmit again.

Once a terminal determines that it has authority to access the bus, it transmits its current message. Each message sent across the bus by a terminal is prefaced by a 16-bit label, which usually identifies the type of data contained in the message. If a another terminal is programmed to receive the same label, it and others so programmed collect the data following the label and send it to their attached subsystem. A bus terminal may also be programmed to provide an Interrupt Strobe signal (IS*) upon receipt of a label. After the label is received, the terminal generates an exception processing vector from its own memory and provides that vector and the IS* signal to the subsystem, most likely a computer, to process the interrupt.

To guarantee that more than two systems may harmoniously share the bus, a number of operational modes have been developed: Mode A,
Mode B, Mode C, and SCAT-C. In Mode A, a terminal is allowed to transmit on the bus if 1) its own gap timer has reached its terminal count, indicating that it has the privilege to use the bus and 2) a second timer, called an Transmit Interval timer (TI), has also elapsed. The TI timer is to indicate that a period of time has passed which is sufficiently long to have allowed all other devices access to the bus. To ensure that all terminals have had the opportunity to transmit, all transmitting terminals are required to broadcast a message when their turn arrives. In this mode, all transmissions from all terminals are specified to be of fixed length and will situate themselves into a fixed order for transmitting, usually based upon the increasing length of their gap timers.

One complete set of transmissions from all bus terminals is called a frame. Under Mode A, a transmission from a given terminal occurs at the same position among the messages of all the other terminals. Given the periodicity of the transmissions, all frames are identical in duration and message length.

Mode A is illustrated in figure 2.4. The first bus terminal to transmit has had both its TI and TG timers expire, thus granting it access to the bus. Once a transmission is concluded, the terminal's interval timer is reloaded and must expire before the gap timer begins to count. This gives the other bus terminals an opportunity to access the bus while preventing the terminal with the shortest gap timer value from monopolizing the bus. Subsequent terminals then transmit, following in order of increasing gap timer values.
Figure 2.4. DATAC Operation - Mode A
Mode B is similar to Mode A, except that upon detecting "bus quiet time," each terminal must first count to a common value, called a Synchronization Gap or Sync Gap (SG) value, before a terminal may enable its own unique TG timer. The SG value is typically shorter than the TI value of Mode A but longer than the longest TG value. Expiration of this counter explicitly signals the beginning of a frame to the terminals. By analogy to the monopolization avoidance mechanism mentioned above, under mode B a terminal may not transmit again until both the TG and SG timers have expired.

At first, the use of an additional counter might seem to complicate the terminal access protocol since bus monopolization is capably handled by the TI timer. However, by using the Synchronization Gap, a bus terminal may vary the length of the messages it sends across the bus from frame to frame without disrupting the other terminals. The availability of an explicit signal to indicate when all terminals have had the opportunity to transmit allows a terminal to begin the access arbitration process at the earliest possible moment. More of the bus's active state is spent transmitting useful data instead of sending stale or redundant data simply to provide timing information. An alternative perspective is to treat the expiration of the SG timer as a signal that all terminals have had an opportunity to access the bus. Mode B is shown pictorially in figure 2.5.

Since a set of transmissions from each terminal comprises a frame, we can see that one frame will likely be different from its predecessor due the possibility of variable message lengths.
CRITERIA FOR TRANSMISSION FROM TERMINAL X:

1. SYNCHRONIZATION GAP HAS EXPIRED
2. TERMINAL GAP Gx SUBSEQUENTLY EXPIRES BEFORE ANY OTHER TERMINAL GAP

A SYNCHRONIZATION GAP LARGER THAN ANY TERMINAL GAP PRECEDES EACH TRANSMISSION FRAME. AFTER THE SYNC GAP HAS EXPIRED, THE GAP TIMERS BEGIN COUNTING TO THEIR UNIQUE VALUES. THE SHORTEST GAP EXPIRES FIRST, ENABLING TRANSMISSION FROM THAT TERMINAL. UPON COMPLETION OF THE TRANSMISSION, THE MOST RECENTLY EXPIRED GAP COUNTER IS INHIBITED UNTIL THE BUS BECOMES BUSY FROM THE NEXT GAP TIMER TO EXPIRE, THUS PREVENTING ANY ONE TERMINAL FROM MONOPOLIZING THE BUS.

Figure 2.5. DATAC Operation - Mode B
However, at some point each terminal will establish a pattern to its message lengths. When an observable pattern of transmissions emerges, it is labeled a Major Frame (MAF). A Major Frame is a collection of unique Minor Frames (MIF). Thus, Mode B is periodic over its Major Frames, but aperiodic over Minor Frames. This method is more efficient than Mode A because the bus need not be preoccupied with sending redundant or extraneous data simply to ensure that the message lengths are identical from frame to frame. Transmissions contain only new data, which allows for shorter message lengths. The access arbitration process may begin sooner, since an arbitrary waiting period is no longer required.

With frame timing information explicitly available via the SG timer, a terminal can be programmed not to transmit during certain frames without affecting the other terminals. This allows for improved bus expansion capability over Mode A. Since only essential data appears on the bus, there is more quiet-time available as expansion space for new terminals to use for transmission.

Mode C is a variation on Mode B; it uses the same access protocol and synchronization concepts. The difference lies in the source of the Synchronization Gap information. Under Mode B, synchronization occurs when the Sync Gap count is reached. Under Mode C, synchronization is achieved by using an external event to alert the bus terminals that a new minor frame is beginning. A separate timer or pulse generator programmed for the minor frame repetition rate would be a simple example of such an external event.

The SCAT-C mode in use aboard the ATOPS is a specific
implementation of Mode C. SCAT is an acronym for System-Clocked Autonomous Terminal. This name suggests that the frame clocking or timing information is available from the DATAC system itself, rather than relying on some external device. Just as Mode B provides for synchronization of the frames by use of the SG timer, SCAT-C provides for synchronization by using the terminal's ability to produce an interrupt on reception of a label. This interrupt signals to the bus terminals that a new frame is to begin.

Since the clocks in the bus terminals will drift, eventually the terminal gap values could overlap, resulting in periods of increased access contention. By providing explicit synchronization information at the beginning of every frame, the various timers are coordinated and the contention problem minimized. One terminal on the bus is given the additional housekeeping task of transmitting the SCAT label, either by itself as a dummy message or as the preface to a data message. This terminal is known as the SCAT terminal. The purpose of this label is to produce an interrupt in all DATAC terminals which is the signal to synchronize their individual TG timers. This synchronization process occurs for all frames. By not compelling each system to wait a fixed amount of time merely for coordination purposes, this explicit synchronization operation allows the terminals to access the bus in a more efficient manner while avoiding access contention problems caused by the drift of the terminals' local clocks.

Operation under the SCAT-C mode is shown in figure 2.6. Major Frame Synchronization (MAPS) is produced when N MIF synchronization
CRITERIA FOR Tx FROM TERMINAL x:

1.) SCAT pulse has been processed
2.) TERMINAL GAP has expired
3.) TERMINAL is scheduled to Tx in the current minor frame

- Message duration may vary from Minor Frame to Minor Frame.
- All Minor Frames are defined to be 10 ms in duration.
- Currently, terminals transmit either once every MiF or MaF

WHERE:

<table>
<thead>
<tr>
<th>Minor Frame</th>
<th>Messages from Terminals</th>
<th>Minor Frame</th>
<th>Terminal</th>
<th>Gap Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>M1</td>
<td>M2</td>
<td>M3</td>
<td>M4</td>
<td>M5</td>
</tr>
<tr>
<td>M1, M2, M3, M4, M5</td>
<td>1,2,3,...,5</td>
<td>TG1</td>
<td>TG2</td>
<td>TG3, TG4, TG5</td>
</tr>
<tr>
<td>M1, M2, M3, M4, M5</td>
<td>1,2,3,...,5</td>
<td>TG1</td>
<td>TG2</td>
<td>TG3, TG4, TG5</td>
</tr>
<tr>
<td>M1, M2, M3, M4, M5</td>
<td>1,2,3,...,5</td>
<td>TG1</td>
<td>TG2</td>
<td>TG3, TG4, TG5</td>
</tr>
<tr>
<td>M1, M2, M3, M4, M5</td>
<td>1,2,3,...,5</td>
<td>TG1</td>
<td>TG2</td>
<td>TG3, TG4, TG5</td>
</tr>
<tr>
<td>M1, M2, M3, M4, M5</td>
<td>1,2,3,...,5</td>
<td>TG1</td>
<td>TG2</td>
<td>TG3, TG4, TG5</td>
</tr>
</tbody>
</table>

Figure 2.6. DATAC Operation - Mode C (SCAT-C Type)
marks have been counted by each terminal, where N is the Major/Minor Frame ratio. Subsequent MIF synchronization marks determine which Minor Frame (0 - N-1) is in progress. The Major Frame Synchronization signal can be thought of as the carry output of a modulo-N counter and the Minor Frame number as the output of the same counter. Depending on the MIF number, the terminal may elect to not transmit. Under SCAT-C operation, a terminal transmits either once every MIF or every MAF using a variable length message like that used under Mode B.

It is significant to note that the demise of the bus terminal assigned to provide the SCAT label would constitute a central failure. Immunity to central failure is one of DATAC's principal design features. To circumvent this failure mode, the DATAC terminals incorporate a series of checks which ensure that all bus terminals conform to the SCAT-C protocol. Should one of these checks fail, each terminal would make the identical assumption that the SCAT terminal has failed to provide the needed system clocking information and would then revert to Mode A of operation.

2. DATAC Capabilities. Currently, the DATAC system offers these capabilities: 127 terminals can use a given bus, each terminal either reads data from or puts data to the bus; a data transfer rate of 1 megabit per second; a 16-bit address field associated with each word in a message which provides for up to 65,536 data words per subsystem; three modes of operation varying from completely periodic (Mode A) to periodic only over a Major Frame (SCAT-C Mode). DATAC capabilities are summarized in table 2.1 [9].
### Table 2.1. Summary of DATAC System Capabilities

<table>
<thead>
<tr>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATA RATE</td>
<td>1 MEGABIT PER SECOND</td>
</tr>
<tr>
<td>NUMBER OF TERMINALS</td>
<td>LOGICAL LIMIT OF 127</td>
</tr>
<tr>
<td>NUMBER OF SUB-SYSTEMS PER TERMINAL</td>
<td>16-BIT ADDRESS FIELD</td>
</tr>
<tr>
<td>TERMINAL CONFIGURATION</td>
<td>SIMPLEX</td>
</tr>
<tr>
<td>MODES OF OPERATION</td>
<td>A. TOTALLY PERIODIC</td>
</tr>
<tr>
<td></td>
<td>B. NON-PERIODIC</td>
</tr>
<tr>
<td></td>
<td>C. NON PERIODIC USING MAJOR/MINOR FRAMES</td>
</tr>
<tr>
<td>TRANSMISSION INTERVAL</td>
<td>1 ms TO 28 ms</td>
</tr>
<tr>
<td>SYNC GAP</td>
<td>4 OPTIONS AS MULTIPLE OF BASIC SYNC GAP (16 µs)</td>
</tr>
<tr>
<td>TRANSMISSION GAP</td>
<td>127 OPTIONS AS MULTIPLE OF BASIC TRANSMISSION GAP (2 µs) CONFIRM</td>
</tr>
<tr>
<td>MAJOR/MINOR CYCLE RATIO</td>
<td>MAXIMUM OF 255</td>
</tr>
<tr>
<td>SUB-SYSTEM INTERFACE</td>
<td>MODIFIED Z8000 I/O (Z-BUS SUBSET)</td>
</tr>
<tr>
<td>MAXIMUM MESSAGE LENGTH</td>
<td>31 STRINGS</td>
</tr>
<tr>
<td>STRING LENGTH</td>
<td>LABEL PLUS n DATA WORDS (1 &lt; n &lt; 65535)</td>
</tr>
<tr>
<td>WORD LENGTH</td>
<td>16 BITS</td>
</tr>
</tbody>
</table>
3. An Overview of Two Popular Aircraft Data Busses. DATAC was designed with commercial aviation applications in mind. However, there are two other common aircraft busses currently in use which are similar in concept to DATAC. These are MIL STD 1553B and ARINC 429. While all three of these systems provide a similar function (distributing data from one system to another via a common serial bus), the characteristics of the individual systems are quite dissimilar. The following is a brief summary of these two systems as well as a comparison with DATAC:

a. MIL STD 1553B. As its name implies, MIL STD 1553B was originally conceived as a way to replace point-to-point wiring between avionics systems aboard military aircraft [10]. A 1553B implementation consists of the bus medium, a bus controller, and two or more remote terminals (RTs). The 1553B bus employs a shielded, twisted-pair for its bus media with direct coupling from its remote terminals to its subsystems. There are three types of messages which may appear on a 1553B bus: command, status, or data. All message activity is managed by the bus controller, which initiates all bus activity. The presence of a bus controller suggests a central failure mode, which has led to the MIL-STD 1553B specification that an installation be completely redundant. The typical architecture for a 1553B system is shown in figure 2.7; the various word transmission formats appear in figure 2.8.

The data encoding scheme employed by the DATAC terminals to place their messages on the bus is the same as that used by MIL STD 1553B. The Manchester II biphase code uses a return-to-zero (RZ)
Figure 2.7. Typical MIL STD 1553B Installation
Figure 2.8. MIL STD 1553B Command/Data/Status Word Formats
scheme for encoding binary data bits. Unlike the non-return-to-zero (NRZ) coding of the popular EIA RS-232C standard [11], a bit state is determined by a positive-to-negative or negative-to-positive transition, relative to a cycle of the bit clock. A logic "1" is defined as a positive-to-negative excursion, while a logic "0" is a negative-to-positive transition. Figure 2.9 compares the coding differences between a serial data stream encoded in an NRZ scheme like RS-232 and the equivalent data using the Manchester II encoding.

A typical exchange begins with the bus controller (BC) sending a command message to a remote terminal (RT), perhaps a command to receive data. The RT replies with a status message, indicating that the command was received and understood. The BC then addresses a second RT, instructing it to transmit data. The sender RT then sends a status message to acknowledge that the command was received correctly and begins to send data across the bus. Because of the broadcast nature of the bus, any of the remote terminals on the bus is capable of receiving the data which appears on the bus; however, only the RT commanded to receive the data by the BC will actually process it.

After the exchange between the listener RT and the talker RT is complete, the listener sends a status message to the current bus controller and indicates whether the message transfer was successful. The controller then either proceeds to the next transaction if the transmission proceeded correctly or it may retry the transfer. If the transfer cannot be completed correctly in a given number of attempts, the controller may order the errant system to shut down.
Figure 2.9. Manchester II Data Encoding
In addition to the talk, listen, and shut-down commands, 1553B supports a variety of other commands; some of these are listed in table 2.2. A summary of MIL STD 1553B's features appear in table 2.3.

b. ARINC 429. ARINC 429, also known as the Mark 33 Digital Information Transfer System (DITS) is a bus specification sponsored by Aeronautical Radio, Incorporated [12]. Like DATAC and MIL STD 1553B, it also uses a shielded, twisted pair as its bus medium. ARINC 429 is intended as a method of distributing digital information from a single data source to multiple receivers which have need of the same data. The DITS specification does not allow for bidirectional transfer of data on a single bus. The implication is that for every data source, there will be at least one bus to service that source. If a system which transmits data is also a data sink, by definition another bus would be required to provide data to that system. Further, if a given system needed data from more than one source, it would require multiple busses to channel that data, as well as a means to arbitrate between the two or more sources of incoming information.

ARINC 429 provides for the transmission of Binary Coded Decimal (BCD), two's complement fractional binary notation (BNR), discrete-coded or ASCII-character information. Provisions are made for two data rates: a "high" rate of 100 kilobits per second ± 1% and a "low" data rate of 12 to 14.5 kilobits per second ± 1%.

A message from a ARINC 429 source consists of 32-bit words in the format shown in figure 2.10. An 8-bit label field specifies the
Table 2.2. MIL STD 1553B Bus Controller Commands

<table>
<thead>
<tr>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dynamic Bus Control</td>
</tr>
<tr>
<td>Synchronize</td>
</tr>
<tr>
<td>Transmit Status Word</td>
</tr>
<tr>
<td>Initiate Self-Test</td>
</tr>
<tr>
<td>Transmitter Shutdown</td>
</tr>
<tr>
<td>Override Transmitter Shutdown</td>
</tr>
<tr>
<td>Inhibit Terminal Flag Bit</td>
</tr>
<tr>
<td>Override Inhibit Flag Bit</td>
</tr>
<tr>
<td>Rest Remote Terminal</td>
</tr>
<tr>
<td>Transmit Vector Word</td>
</tr>
<tr>
<td>Transmit Last Command</td>
</tr>
<tr>
<td>Transmit Bit Word</td>
</tr>
<tr>
<td>Selected Transmitter Shutdown</td>
</tr>
<tr>
<td>Override Selected Transmitter Shutdown</td>
</tr>
</tbody>
</table>
Table 2.3. Summary of MIL STD 1553B Features

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>APPLICATIONS</td>
<td>DoD AVIONICS</td>
</tr>
<tr>
<td>DATA RATE</td>
<td>1 MHz</td>
</tr>
<tr>
<td>WORD LENGTH</td>
<td>20 BITS</td>
</tr>
<tr>
<td>NUMBER OF DATA BITS PER WORD</td>
<td>16 BITS</td>
</tr>
<tr>
<td>TRANSMISSION TECHNIQUE</td>
<td>SIMPLEX</td>
</tr>
<tr>
<td>OPERATION</td>
<td>ASYNCHRONOUS</td>
</tr>
<tr>
<td>ENCODING</td>
<td>MANCHESTER II BIPHASE</td>
</tr>
<tr>
<td>SIGNALING FORMAT</td>
<td>BIPOLAR VOLTAGE</td>
</tr>
<tr>
<td>BUS COUPLING</td>
<td>TRANSFORMER</td>
</tr>
<tr>
<td>BUS CONTROL</td>
<td>SINGLE OR MULTIPLE UNITS</td>
</tr>
<tr>
<td>TRANSMISSION MEDIA</td>
<td>TWISTER PAIR, SHIELDED</td>
</tr>
</tbody>
</table>
1. GENERAL WORD FORMATS

<table>
<thead>
<tr>
<th>P</th>
<th>SSM</th>
<th>BCD MSC</th>
<th>BCD CH #2</th>
<th>BCD CH #3</th>
<th>BCD CH #4</th>
<th>BCD CH #5</th>
<th>SDI</th>
<th>LABEL</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**EXAMPLE**

<table>
<thead>
<tr>
<th>P</th>
<th>SSM</th>
<th>BCD MSC</th>
<th>BCD CH #2</th>
<th>BCD CH #3</th>
<th>BCD CH #4</th>
<th>BCD CH #5</th>
<th>SDI</th>
<th>LABEL</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**BCD WORD FORMAT EXAMPLE (NO DISCRETES)**

<table>
<thead>
<tr>
<th>P</th>
<th>SSM</th>
<th>BCD MSC</th>
<th>BCD CH #2</th>
<th>BCD CH #3</th>
<th>BCD CH #4</th>
<th>BCD CH #5</th>
<th>SDI</th>
<th>LABEL</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**GENERALIZED BCD WORD FORMAT**

<table>
<thead>
<tr>
<th>P</th>
<th>SSM</th>
<th>BCD MSC</th>
<th>BCD CH #2</th>
<th>BCD CH #3</th>
<th>BCD CH #4</th>
<th>BCD CH #5</th>
<th>SDI</th>
<th>LABEL</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**BCD WORD FORMAT EXAMPLE (NO DISCRETES)**

<table>
<thead>
<tr>
<th>P</th>
<th>SSM</th>
<th>BCD MSC</th>
<th>BCD CH #2</th>
<th>BCD CH #3</th>
<th>BCD CH #4</th>
<th>BCD CH #5</th>
<th>SDI</th>
<th>LABEL</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**GENERALIZED BNR WORD FORMAT**

<table>
<thead>
<tr>
<th>P</th>
<th>SSM</th>
<th>BCD MSC</th>
<th>BCD CH #2</th>
<th>BCD CH #3</th>
<th>BCD CH #4</th>
<th>BCD CH #5</th>
<th>SDI</th>
<th>LABEL</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**BCN WORD FORMAT EXAMPLE (NO DISCRETES)**

<table>
<thead>
<tr>
<th>P</th>
<th>SSM</th>
<th>BCD MSC</th>
<th>BCD CH #2</th>
<th>BCD CH #3</th>
<th>BCD CH #4</th>
<th>BCD CH #5</th>
<th>SDI</th>
<th>LABEL</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**ALPHA/NUMERIC (ISO ALPHABET NO.5) DATA - INITIAL WORD FORMAT**

<table>
<thead>
<tr>
<th>P</th>
<th>SSM</th>
<th>DATA CH#3</th>
<th>DATA CH#2</th>
<th>DATA CH#1</th>
<th>LABEL</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**ALPHA/NUMERIC (ISO ALPHABET NO.5) DATA - INTERMEDIATE WORD FORMAT**

<table>
<thead>
<tr>
<th>P</th>
<th>SSM</th>
<th>DATA CH#3</th>
<th>DATA CH#2</th>
<th>DATA CH#1</th>
<th>LABEL</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**ALPHA/NUMERIC (ISO ALPHABET NO.5) DATA - FINAL WORD FORMAT**

(Taken together, above three word format examples show encoding of the word (ALPHA)

P - Odd Parity Bit
SSM - Sign/Status Matrix
Pad - Unused data bits set to a "0" or Specified Pad Value
SAI - Source, Destination Identifier

Figure 2.10. General Word Formats and Encoding Examples for ARINC 429
nature of the data contained in the message. Two source/destination bits are used to clarify the source or destination of the data if a) it originated from a system having more than one data source or b) the target is one of several systems served by one bus interface unit. Nineteen bits are available for data, either BCD, BNR, discrete-coded, or ASCII-character. Two bits, called the sign/status matrix bits, are available for providing dimensional units to the data: either PLUS, NORTH, EAST, RIGHT or TO indications, or MINUS, SOUTH, WEST, LEFT, or FROM may be coded. The most significant bit of the data word is always reserved for odd parity.

A three-value bipolar RZ coding scheme is used for translating the binary data. The three levels of the coding scheme are high, 10 \( V \pm 1/2 \), low, \(-10V \pm 1/2\), and null, \( 0V \pm 1/2 \). Figure 2.11 illustrates valid signal levels for the DITS 33 bus.

The two binary logic states are represented by a sustained null-to-high or null-to-low transition for one-half of the specified bit-time. We may illustrate using the timing of the high speed mode: a logic "1" is represented by a transition from the null level to high holding high for 5 \( \mu \)s and then returning to the null value for 5 \( \mu \)s. Similarly, for the logic "0" case, the coding is a null to low level for 5 \( \mu \)s and then returning to the null level for 5 \( \mu \)s. A comparison of this coding scheme with the NRZ coding used by RS-232C is shown in figure 2.12. A summary of ARINC 429 features is given in table 2.4.

c. A Comparison of DATAC, ARINC 429, and MIL STD 1553B. It is DATAC's well-behaved sharing of a common bus that leads
Figure 2.11. Voltage Levels for ARINC 429 Bit Encoding
Figure 2.12. Data Bit Encoding for ARINC 429 Transmissions Using Bipolar Three-Level RZ Coding
<table>
<thead>
<tr>
<th>APPLICATION</th>
<th>COMMERCIAL AVIONICS</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATA RATE</td>
<td>100 KILOBITS PER SECOND (HIGH SPEED)</td>
</tr>
<tr>
<td></td>
<td>12.0 - 14.5 KILOBITS PER SECOND (LOW SPEED)</td>
</tr>
<tr>
<td>WORD LENGTH</td>
<td>32 BITS</td>
</tr>
<tr>
<td>DATA BITS PER WORD</td>
<td>19</td>
</tr>
<tr>
<td>TRANSMISSION</td>
<td>UNIDIRECTIONAL (PER BUS)</td>
</tr>
<tr>
<td>ENCODING</td>
<td>BIPOLAR RETURN-TO-ZERO (RZ)</td>
</tr>
<tr>
<td>BUS COUPLING</td>
<td>DIRECT-CONNECT OR OPTO-ISOLATED</td>
</tr>
<tr>
<td>TRANSMISSION MEDIUM</td>
<td>TWISTED-PAIR, SHIELDED</td>
</tr>
</tbody>
</table>
to its advantages over other airborne data bus systems. An obvious advantage is that of weight savings. By providing a means for numerous flight-management and control systems to communicate without the need for point-to-point wiring, the mass of separate wiring harnesses is eliminated. DATAC requires no central controller to arbitrate bus activity like that used by MIL STD 1553B. The absence of a unit whose purpose is to manage the bus results in improved reliability due to reduced system component count and, perhaps most importantly for a commercial aircraft, it removes a source of central failure. This immunity to central failure is a crucial advantage for commercial systems seeking FAA certification. It also eliminates the need for a redundant or backup system.

Another DATAC advantage is provided by the use of a twisted pair for the bus media and inductive couplers as a means to attach the bus terminals to the bus. Since there is no invasive physical connection made when adding a terminal to the bus, one may add or delete terminals and the associated subsystems merely by clamping or unclamping the bus coupler. Associated with the physical ease of adding and deleting systems from the bus configuration is that the broadcast nature of the bus makes the addition and deletion of systems transparent to the existing systems. A given subsystem and its terminal transmit data via the bus, identifying the transmission with a unique label. Other systems on the bus receive this data only if they are programmed to receive this label; the originating system has no knowledge about the other systems that use the bus and relies on the protocol management features of its terminal to ensure that
its transmissions and receptions are sound. Thus, a system and its
terminal may be added or deleted without modifying other systems and
terminals on the bus. This feature makes the bus very flexible and
allows for easy field maintenance of the system.

Thus while all three of these systems share a common purpose,
that of minimizing or eliminating point-to-point wiring among
aircraft systems, the means by which they implement their respective
systems are quite different. Table 2.5 summarizes comparative
features of the three systems and points out some of their
contrasting design elements.

B. The DATAC Bus Monitor

While high reliability, reduced weight, ease of certification,
and simplified maintenance are factors in DATAC's favor, there is a
liability in its autonomous access protocol. Because each bus
terminal and therefore subsystem uses the bus independently of the
other systems, there is no easy way to interact with those systems
for diagnostic purposes. For example, in a development phase it
might be desired to interrogate a specific system to determine if it
is operating properly. However, this is not possible because of the
way DATAC is designed: the bus' broadcast protocol implies that there
is no requisition of a transmission from one system to another. That
is, if subsystem A expects to receive information from system B, then
A must wait for B to transmit. Unless A is programmed with the time
frame in which B is to operate (which suggests that system A has been
prepared for a specific bus architecture and thus is skirting the
transparent system addition/deletion feature of the system), system A
Table 2.5. Summary of DATAC, ARINC 429 and MIL STD 1553B Features

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>M</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>TWISTED-PAIR AT THE BUS MEDIA (SHIELDED CABLE FOR VOLTAGE SIGNALING)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>BIPOLAR VOLTAGE SIGNALING USED FOR DATA TRANSMISSION*</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>&quot;STUBS&quot; PHYSICALLY ATTACHED TO BUS MEDIA**</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>INDUCTIVE COUPLERS CLAMP ONTO BUS MEDIA (REMOVABLE)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>MULTIPLE MESSAGE FORMATS (COMMAND, STATUS, DATA)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SINGLE MESSAGE FORMAT (DATA ONLY)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>BIDIRECTION DATA TRANSFERS BETWEEN SYSTEMS</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SINGLE BUS PER TRANSMITTER</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>&quot;HIGH-SPEED&quot; OPERATION (1 Mbps FOR 1553B AND DATAC, 100 kbps FOR 429)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>MANCHESTER BIPHASE NRZ ENCODING</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>BIPOLAR RZ ENCODING USED</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>16-BIT ADDRESSES/IDENTIFIERS USED PER DATA MESSAGE</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>8-BIT LABELS USED PER DATA MESSAGE (2 EXTRA BITS AVAILABLE FOR ROUTING INFORMATION)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**LEGEND**

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>M</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ARINC 429</td>
<td>MIL STD 1553B</td>
<td>DATAC</td>
</tr>
</tbody>
</table>

*DATAC ALLOWS THE USE OF VOLTAGE, CURRENT OR OPTICAL SIGNALING
**DATAC OPTICAL-FIBER SYSTEMS REQUIRE DIRECT CONNECTION TO THE BUS MEDIUM
would ordinarily have no knowledge of system B's transmission schedule and might either process increasingly stale data or, more likely, wait forever for the data to appear.

Alternatively, it could be useful to have a second system able to reconstitute bus data aside from the intended "destination."

Consider this bus scenario: if an instrument which relies on bus data for its display behaves erratically, one might suspect the display, the subsystem which services the display, the subsystem's bus terminal, the sending subsystem, or the sending subsystem's bus terminal. In a normal flight configuration, it is possible that there is another system which uses the same data as the errant display; this system could be used in an attempt to correlate values and locate the error source. If another system did not use the same data, it would be impossible to determine the error source, short of bringing the system down for maintenance and scrutinizing the system with tools to specifically monitor the data flow. It is this need for a diagnostic tool that has led to the development of the Bus Monitor.

1. **Need for the Bus Monitor.** In general terms, the Bus Monitor allows an operator to monitor the data flow across the bus and provide a means to correlate the data appearing on the bus with the data in use by both the sender and receivers. Put simply, the Bus Monitor is intended as a diagnostic tool to provide a means to overcome constraints imposed by the system's desirable features, constraints which make monitoring this experimental system difficult.

2. **Proposed Bus Monitor Functions.** While the basic Bus
Monitor will simply receive and display data from the DATAC bus, greater utility lies in its ability to perform some kind of operation on the data. It is desirable for the Bus Monitor to be able to provide these functions:

a. **Data Stream Display**. In this mode of operation, the data stream from the DATAC bus is sent to the Bus Monitor console unit for display and optional storage. The data from the bus is in hexadecimal form and is displayed as such in tabular form on the console computer's display. This mode of operation is intended primarily for laboratory use, where monitoring the actual transfer of data from system to system is desired. Two submodes of operation are 1) stream all data as it appears on the bus (for example, display an entire Major- or Minor-Frame Transmission, depending on the operational mode) or 2) specify a label, which allows the operator to view activity from a single source.

b. **Data Stream Interpretation**. Here, the Bus Monitor processor receives and converts the bus data into real-world parameters. The principal use for this mode of operation is to provide a facility for locating potential sources of data errors in bus systems by providing an independent means of data evaluation.

If it is essential that all data appearing under a label be converted, it may be necessary to specify an acceptable skip rate on the appearance of labels. This would result in a two-of-three or three-of-four type display of a particular system's data stream. Since the DATAC update rate is more than adequate to provide the 5 to 50 Hz bandwidth offered by ARINC 429, depending upon the mode of
operation, a message skip may not be noticeable.

c. Echo-check. This mode of operation is intended to allow the Bus Monitor operator to do that which the broadcast protocol does not allow: interrogation of a specific subsystem. The DATAC terminals are designed so that they listen to their own transmissions and attempt to detect and correct a known transmission error. In this way, raw transmission errors are virtually eliminated. However, if either the originating subsystem's or the receiving system's bus terminal is defective, the terminal may incorrectly encode or decode the data. Should this happen, the data appearing to the receiving subsystem will not be the same as the data from the originator and the bus terminal protocol managers would be unable to detect this condition.

To deal with this possibility, certain features of the DATAC system architecture provide a way of "interrogating" a bus terminal and its subsystem. Given the expansion space in the DATAC system, each bus terminal can be programmed to receive a label sent by the Bus Monitor. The data appearing under this label, essentially a test pattern, is directed to the RAM of the interface shared by the bus terminal and the subsystem that the terminal serves. For most of the current ATOPS systems, this shared RAM is typically a two-port device which the DATAC terminal accesses as another computer system's main memory; the subsystem usually treats the RAM as addresses in its input/output space.

After being deposited into the shared RAM, this same block of data will then be sent out by the target system during a normally
unused time slot under a label that the Bus Monitor's terminal alone is programmed to receive. The Bus Monitor will compare the transmitted values with those received and determine if there is any corruption of data taking place between the subsystem and the terminal.

This mode of operation is rather elegant in that when not in use, it does not use any bus access time. As the bus is expanded and message traffic increases, the test-pattern messages may be shortened in length but increased in number and the transmissions themselves staggered so that every bus terminal-subsystem buffer pair could be tested without significantly affecting traffic on the bus.

d. Transmission Schedule Confirmation. Depending upon the quality of information required from the Bus Monitor, this function could be considered to be least in line with the DATAC of philosophy of subsystem independence. In order to monitor the order of the subsystem's transmissions, it is necessary for the Bus Monitor to be programmed with the current bus configuration. The broadcast protocol of the DATAC bus was chosen so that any given subsystem need not be modified to deal with the existence of another system on the bus. For the Bus Monitor to "know" what other systems are on the bus, it must be programmed with the current bus configuration. This would require modifying the Bus Monitor's bus terminal every time a system is added to or deleted from the bus.

As an alternative, it would be possible for the Bus Monitor to wait for the completion of the first frame under Mode A or the first Major Frame under Modes B/C mode, tabulate the systems which
communicated in that time, and determine if that tabulation conforms to the same schedule obtained by monitoring subsequent frames. This tabulation method is somewhat more in line with the transparent system addition/deletion features of DATA in that the Bus Monitor displays the stations that are broadcasting and has no knowledge of those that are not. In this case, the Bus Monitor operator would be responsible for knowing exactly which systems are using the DATA bus.

Depending upon the system installation and the kind of information desired from the Bus Monitor, it may be best for the operator to allow the Bus Monitor to collect what information it can about the system and then modify the observed system configuration as needed. In this case, no special programming of the Bus Monitor or its bus terminal would be required and the it would still be able to indicate which other systems are correctly using the bus.

With the exception of providing timing information about DATA message transactions, these four modes of operation should be able provide any information regarding DATA bus operations that would be required for diagnostic or troubleshooting purposes.
III. IMPLEMENTATION OF THE BUS MONITOR DEVELOPMENT SYSTEM

Three requirements have been identified for minimal implementation of the Bus Monitor proposed functions:

To receive data from the bus
To identify the origin of the message
To identify data within the stream

The Bus Monitor is a unit that receives all data from the bus and performs various types of analysis on that data. The Bus Monitor requires an interface to act as the liaison between the bus and the monitor unit. The Bus Monitor's communications needs are philosophically simpler than those of a conventional bus subsystem, but its control requirements are more complex. For all operational modes, the DATAC terminal servicing the Bus Monitor system simply will be programmed to receive all messages appearing from the bus. Because of the intricacies of interfacing with the bus, it is undesirable to design new circuitry to provide services similar to those available from existing DATAC bus terminals. Extensions or modifications to a standard bus terminal should be considered only when a terminal does not supply a feature that the Bus Monitor requires.

The first requirement to be addressed is a means to interface the DATAC terminal to the Bus Monitor. Originally, the S100 bus was chosen for the Bus Monitor system (c. 1983) because of the wide availability of hardware support for that environment. The IEEE-696 standardization of this bus [13] extends the bus definition to include 16-bit data transfers as well as providing for a 24-bit...
address bus. It was originally believed that this bus would be appropriate for designing a flexible system that initially was only partially defined in terms of its functions and capabilities.

The Zilog Z8000 16-bit processor family was chosen because of its orthogonal architecture and high performance. The particular member of the Z8000 family used in the Bus Monitor is the non-segmented Z8002, which supports a single 64 kilobyte linear address space. An important qualification of the Z8002 is that the subsystem interface (SSI) used on standard DATAC terminals is a subset of the Z8000 family's native interconnection scheme. This scheme is called the Zilog Z8000 BUS or Z-BUS [14]. The subset of the Z-BUS used by the DATAC subsystem interface is described in Appendix A. By taking advantage of this architectural feature, it is hoped to implement a Bus Monitor system which, when distilled to its final form, would employ a Z8000 as the main processor and allow the Z-BUS interface manage the data transfers from the bus terminal in hardware. Thus, any software investment would be portable across successive generations of systems with little or no modification. Any changes that arise from a new hardware configuration would consist primarily of revisions to control routines which manage Z-BUS access by the DATAC terminal.

A problem with using an S100 system is that the S100 bus and Z-BUS are not compatible. A possible solution to the interconnection problem was to use an existing interface unit between the bus terminal and the Bus Monitor processor known as the Shared Interface Ram, or SIR. A block diagram of a SIR appears in figure 3.1. This
Figure 3.1. Block Diagram of Shared Interface RAM (SIR)
The system is more complex than the S100/Z-BUS interface eventually built for the Bus Monitor because the SIR uses the bus terminal's address lines in conjunction with internal bus terminal and other SSI control lines to provide extended control functions. These functions were principally intended to be used with minicomputers and other complex subsystems. The SIR was developed to minimize interfacing problems such as subsystem support for the data flow, interrupt management and control. However, SIR employs the Z-BUS interface on both the terminal and subsystem sides, and this did not remedy the problem of connecting to the S100 system.

While it would have simplified software design to have a separate memory for the data transfers, it was decided to implement a means for the terminal perform a Direct Memory Access (DMA) on the Z8002 system memory. This was done principally to simplify design of the interface. A DMA access is essentially the type of transfer the terminal performs when it handles data for a subsystem which uses a SIR. This DMA design was realized in an interface that would translate the Z-BUS DMA signals coming from the terminal into the same control lines used for the S100 bus DMA operation.

In order to receive all the data appearing on the bus, the terminal servicing the Bus Monitor must be programmed to process every label that is broadcast. This involves modifying the Erasable-Programmable-Read Only Memory (E/P/ROMS) in the bus terminal's protocol managers. These ROMs are said to provide the terminal's personality because they determine its behavior during bus message transmission and reception.
Because the bus terminal data flow to and from the S100 system will impose constraints on the data processing time of the Bus Monitor main processor, some restrictions will be present. One is that the Bus Monitor probably will not be able to process all the data appearing on the bus during a Major Frame. To counter this, it may not be desired to look at messages arriving under every label or at every value under a label. This implies that the Bus Monitor itself must exercise some control over the data flow from its terminal.

Explicit Bus Monitor control of the bus terminal data activity seems to be a promising approach initially because of the programmable personality of the DATAC terminals. However, dynamic modification of the terminal's personality presents some problems. For example, dynamic modification requires that the personality implementation be a two-port RAM with write protection to take the place of the E/P/ROM that is the standard terminal's personality device. Refer to figure 3.2 for a detailed block diagram of a DATAC bus terminal. Note that both the receiver section and the transmitter section have separate personality devices, although the receiver personality is of prime interest for this system. As with any two-port device, an access contention scheme must be devised, which further complicates what is intended to be a simplifying procedure.

An acceptable approach is to let all messages from the bus go to the Bus Monitor and allow it to determine which to process. This could be done using the interrupt capabilities of the DATAC terminal
Figure 3.2. DATAC Terminal Bus Encoder/Decoder Block Diagram  
(Courtesy Smiths Industries, Ltd.)
to provide the Bus Monitor with a means of identifying the incoming transmissions. Recall that a bus terminal has the ability to recognize labels as an interrupting event, secure a 16-bit vector from its personality configuration, and assert the vector and provide an interrupt signal. The Bus Monitor does not make use of this capability as such at this time, although processing interrupting labels would give Bus Monitor the ability to provide timing information about the bus operations. This would be of particular use in obtaining quantitative data on the operation of the SCAT-C mode. The Bus Monitor will instead use this interrupt capability as a signal that a transmission is coming in from the bus and use the 16-bit interrupt vector as an identification tag. This ability to determine the origin of the bus transmissions and selectively accept or ignore them is nearly as flexible as having a dynamic terminal personality, with the exception of the software overhead of managing the transmission identifications. In light of the simplicity of identifying the transmissions by software instead of the hardware modifications to the DATAC terminal's personality, the software approach was adopted.

A. DATAC Bus Terminal Data Stream

The only way for a DATAC terminal to provide information is to have it read data from an active system. Currently, such systems are not commonplace. To overcome this obstacle, it was decided to create a DATAC terminal simulation. Completely modeling the operating characteristics of a terminal is a monumental task, given the vast number of functions that the device is capable of performing.
Instead, it was decided to have a simple microcomputer system providing a data stream and control signal subset, comparable to those provided by a terminal to a subsystem. This simulator would provide a flexible environment in which to develop and test the Bus Monitor's software and then gauge its ability to respond. This configuration allows the timing and data movement capabilities of the Bus Monitor/terminal system to be tested and evaluated.

A 68000-based Single Board Computer was used to serve as the terminal simulator by virtue of its ability to manage data in 16-bit quantities as the bus terminal does. The programmer's model for the MC68000 microprocessor is shown in figure 3.3. The MC68000 supports a single 16 megabyte linear memory address space. Its peripherals are memory-mapped instead of I/O mapped, as is the case for the Intel iAPX-86 and Zilog Z8000 families. Because of its large linear address space, the 68000 lends itself to fast access of large data structures.

The 16 general purpose registers are divided into a data register group and an address register group. The 8 data registers are used primarily for arithmetic operations, while the 8 address registers are used for pointers and indices for addressing operations. The address registers provide for very fast address calculation times for data movement operations. This is an important consideration in the DATAC data stream simulation, given that a new data word, address word, and attendant control signals are required every 20 μs. The particular SBC system in use was selected because of its affordability, software, and expandability. It features two
Figure 3.3. M68000 CPU Programmer's Model
serial ports, a 20-bit parallel I/O device configurable as a parallel printer port, a very complete monitor program including a one-pass assembler, and a wire-wrap area for expansion. A typical installation for the terminal simulator is shown in figure 3.4. A IBM PC acts as the console device for the 68000-based single board computer, providing program storage and I/O facilities. Software is developed on the PC using a cross-assembler and is down-loaded to the simulator using the Motorola S-record format [15]. The expansion facilities of the microcomputer were used to add parallel I/O devices to simulate the terminal subsystem interface, which was then connected to the Bus Monitor computer system.

1. The Subsystem Interface Simulation. To simulate the Subsystem Interface (SSI) found on the DATAC bus terminals, the SBC was fitted with two Rockwell/Synertek 6522 Versatile Interface Adapters (VIAs). Each of these eight-bit parallel I/O devices features two individually addressable eight-bit ports as well as four configurable I/O lines, normally used for handshaking or interrupt handling purposes. Appendix B provides a description of the internal register configuration of the 6522 VIA, as well as a summary of its operational modes.

The 68000 accesses two bytes simultaneously on a even address boundary by employing two control lines to enable access to the upper and lower bytes of the word. These lines are appropriately called Upper Data Strobe (UDS*) and Lower Data Strobe (LDS*). During a memory access, if both lines are asserted, a word operation follows, if either line is asserted (but not both), a byte operation is
Figure 3.4. DATAC Terminal Simulator Implementation
performed. Figure 3.5 illustrates how the 68000 performs word accesses on memory and peripheral devices.

To provide the 16-bit ports needed for the subsystem interface, each 6522 is identically decoded on a given address. One chip is selected by the UDS* signal, the other by the LDS*. On a word transfer, one byte goes to Port A of one device and the other byte to Port A of the companion chip, thus forming the 16-bit port.

While the main attraction of the control lines associated with each port is their ability to handshake and handle interrupts autonomously, this feature was not used. Due to the relatively long interrupt response time and the fact that the Bus Monitor exists only to receive data from that particular bus terminal, the BUSACK* signal from the Bus Monitor is ignored. This signal is used during Z-BUS DMA operations.

2. **Subsystem Interface Driver.** The address/data line portion of the DATAC bus terminal simulator is shown schematically in figure 3.6. The two VIAs are used in tandem to provide 2 16-bit ports. These act as the multiplexed address/data lines and the steady address lines of the DATAC terminal subsystem interface. The devices have a base address of $30000 ("$" is the conventional notation for base 16 numbers in Motorola publications). Each address reference will select the same register in each device. The VIA labeled VIA2 in figure 3.6 occupies the upper half of the address, VIA1 resides on the lower half. This corresponds to byte address $3000x$ for VIA2 and $3000x+1$ for VIA1, where x is always an even number for word-address alignment.
HEX WORD VALUE $1234:

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1001</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1002</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 3.5. Byte/Word Addressing in the M68000**

- **HIGH ORDER BYTE MEMORY**
  - Even address
  - UDS
  - D8-D15
  - Address decode

- **LOW ORDER BYTE MEMORY**
  - Odd address
  - LDS
  - DO-D7

both bytes are accessed (word operation) when UDS and LDS are asserted
Figure 3.6. Subsystem Interface for DATAC Terminal Simulator
Address/Data/Interrupt Line Generation
Because the SSI simulation uses two identical devices in parallel, a convention needs to be established for managing two sets of identical abbreviations. To clarify which device is being referenced, the following nomenclature is used: a given line from the 6522s will be referred to as \(X_nY_m\), where \(X\) is the type of line: Port (P) or Control (C); \(n\) is the device number (1 or 2); \(Y\) is the port associated with the line (A or B); and \(m\) is the number of the line (0-7 for the port lines, 1 or 2 for the control lines). For example, P1A5 refers to port A, line 5 on VIA1; C2B3 refers to control line CB3 on VIA2. For this implementation, since it is desired to treat this system as a single 16-bit peripheral, subsequent references to Port A or any of the 8-bit registers will refer to the corresponding 16-bit tandem quantity.

In this dual-device configuration, Port A ($30002) provides the steady address information, Port B ($30000) provides the multiplexed address/data bus. Figure 3.7 provides the circuitry which implements the SSI control lines. The control lines used in the SSI simulation are generated using existing address decoding present on the SBC. A J-K flip-flop is used to set and clear the line assigned to the BUSREQ* function. The J-line is decoded at $20000 and the K-line at $40000. Any memory reference to $20000 sets the flip-flop and asserts BUSREQ* and a subsequent reference to $40000 clears the device, deasserting that signal. Since the IS*, WR*, and DS* lines only require pulses of a given duration, these signals were generated using 74LS123 one-shots triggered by address decoding. IS* is triggered by decoding for address $50000 with a pulse width of 20 ns.
Figure 3.7. Subsystem Interface Simulation Control Line Generation
A 40 ns DS* pulse is produced by referencing $60000$ and WR* lasts 50 ns when $70000$ is addressed [16].

The principal reason that the control lines available on the 6522s were not used was that the BSET and BCLR operations needed to set and clear the bits in the 6522 Auxiliary Control Registers to operate the corresponding CAx and CBx lines were too slow. Setting or clearing a bit using those instructions requires at least 12 processor clock cycles (2.5 μs using a 4 MHz clock). By comparison, it is possible to generate an address in 2 μs by using a MOVE Dn,(An) instruction [17]. The 20% savings in signal line management was crucial in keeping the SSI simulation timing as close an actual terminal as possible, given the extensive use of the simulated control lines. By using address decoding for input signals and JK flip-flops and one-shots to provide the proper signal duration and timing, it was possible to produce a bus terminal SSI simulation with a word transfer time of 24.5 μs and a label generation time of 30 μs. This is not as fast as an actual terminal; however, it will be shown that simulated versus "real-time" message times compare favorably.

3. Bus Terminal Data Stream Simulation. The DATAC terminal simulation can be divided into three parts: hardware initialization, data table initialization, and data stream simulation. Hardware initialization covers setting of the parallel ports, timer count value setup and other external event/interface functions. Data table setup organizes the SSI data table which contains entries for word values, load addresses and message lengths. Data stream simulation reads the data table and presents the entries
to the SSI simulation in a near-duplication of a real DATAC terminal subsystem interface.

Appendix C lists the DTERM3 68000 assembly language program which provides the SSI simulation described below. The simulations begin by setting the 68230 free-run counter to provide a level 4 autovector [18, 19] every 50 ms. This is used to coordinate the message transfers to the DATAC Mode A 50 ms frame repetition rate. The 16-bit A and B ports of the tandem 6522 VIAs are configured as output ports to present data words and load addresses to the Bus Monitor Z-BUS/S100 interface card.

Next the user is presented with a "A>" prompt to enter the number of words to appear under each label for this test run. For example, if a 7 were entered, then the simulation run would consist of 704 frames of 5 systems, each sending 7 words per message. With the words-per-message value available, the data table was then built in RAM. Given the data tabling technique used, there was sufficient memory space to store only 5 * 704 or 3502 separate message entries. Each message was summarized by a data table entry consisting of 4 words: the ID tag, corresponding to the bus terminal interrupt vector; the data word count value, which corresponds to a word-organized memory address used by the SIRs; the word-count for this message, used as a loop counter for the transfer routine; and the actual data word used in the transfer. Five entries were made for each frame to simulate the 5 systems broadcasting per frame.

After five entries were made for each frame, a $FFFF was written as an end-of-frame sentinel. A sentinel value was chosen
instead of a nested loop construct for determining end-of-frame. The reason for this was that all of the data registers were in use for the word-transfer operation and thus were unavailable for use as fast-access counters. Due to the long access times associated with memory locations, RAM variables were similarly ineligible. Best performance was obtained by simply checking the ID value to determine an end-of-frame condition. Since an ID/label word takes 4 μs longer to process than a data word, the additional time was used to check the dummy ID value used as the end-of-flag value. Using the extra processing time associated with the ID/label word to cover the overhead of identifying the sentinel value was instrumental in providing as accurate a bus terminal interface simulation as possible.

Once all 3520 data table entries are made, the SSI data transfer simulation starts. Address register A6 is loaded with the starting address of the first table entry containing the 4 data words. After each entry is read, A6 is autoincremented by two so that it is always pointing at the next word to be read. The first word read is the ID tag value, which is saved in D4. The next table entry is the word count/load address, which is saved in data register D3 for easy incrementing. The third entry in the table is the number of words in the current message. This is used internally to the simulation as a loop counter value and is saved in D5. Next, the data word is read into D2 and the word transfer process begins. The ID tag is put on the simulated 16-bit vectored interrupt/address (VI/AD) lines of the SSI. The IS* line then is strobed, latching the
ID tag in the Z-BUS/S100 interface and producing an NMI in the Z8002 system, alerting it that a message is inbound. The word count/load address is placed on the AD/VI lines and then the data word is put on the address/data (A/D) lines of the SSI. The JK flip-flop is set, asserting BUSREQ*. The bus terminal simulation then "assumes" correctly that the S100 bus will be granted to it and proceeds to perform the word-write operation.

With the data word and word-count/load-address already on the appropriate SSI lines, the WR* strobe is asserted and the DS* signal follows. The two strobes overlap by about 20 ns, which conforms to the Z-BUS memory-write timing specification [20] The actual word-write takes place on the rising edge of the DS* strobe.

After a word has been deposited into Z8002 memory, BUSREQ* is released. The register D3 containing the load-address is incremented by one to reference the next word load address and the bus-acquisition, word-write, bus-release procedure is repeated until all of the words appearing under the current ID tag (which are identical for a given frame) have been deposited.

Once a station's message transfer has been simulated, the pointer into the data table points at the ID tag of the next system message to be simulated. The transfer process is repeated until all 5 messages for a given frame have been sent. After a complete frame has been simulated, the message-transfer routine encounters a $FFFFF sentinel value. The 68000 then fetches the data words to simulate the first message of the next frame and waits for the 50 ms timer to elapse, indicating the beginning of a new frame. When all 704 frames
have been simulated, the program jumps back to the user-prompt routine so that another test run can be performed or the program can be cancelled by simply hitting the RETURN key on a null line.

B. Z8000/PC Processing Configuration

The current version of the DATAC Bus Monitor consists of a Z8002-based S100 bus microcomputer which interfaces to the DATAC terminal and an IBM Personal Computer (PC) acting as the Bus Monitor console device. See figure 3.8 for the current laboratory Bus Monitor development system.

The programmer's model for the Z8002 CPU appears in figure 3.9. The Z8002 features a register file of 16 general purpose registers which may be used as accumulators, address registers, or index registers. This member of the Z8000 family features a 64k byte linear address space and a relocatable program status area, which defines where in its main memory the exception processing vectors are located.

The IBM PC serves as a console device to both the Z8002 and 68000 systems. This console system provides program development, peripheral management, and data reduction, display and storage. The PC is equipped with a Sweet Micro Systems Trump Card. The Trump Card is a Z8001-based coprocessor card which provides a Z8000-family assembler, and incremental BASIC and C compilers. The Z8001 is very similar the Z8002 with the exception that it supports a segmented address space [21] of 8 megabytes. When operating in non-segmented mode, the Z8001 is constrained to use a single 64 kilobyte segment and functions identically to its sibling. The internal organization
Figure 3.8. Bus Monitor Software Test System
Figure 3.9. Z8002 CPU Programmer's Model
of the Z8001 is illustrated in figure 3.10.

Programs for the Bus Monitor are written on the PC using screen editors under the MS/PC-DOS environment and then reduced to machine language by the Trump Card's Y assembler [22]. A binary image of the machine language program is then translated to ASCII for downloading to the S100 system. The Trump Card binary file format is illustrated in figure 3.11.

1. File Transfer Utilities. All Bus Monitor software must reside on the 64k memory space of the Z8002; therefore when converting the binary file, any segmentation information produced by the Y assembler is ignored. The information from the file header section used in the binary-to-ASCII transformation is shown highlighted. Prior to downloading, the program file is converted into one of two formats: standard TekHex [23], a record-oriented format similar to Motorola's S-record format, or an ASCII file which embeds commands of the MPU8000 II monitor to load the program one word at a time. Appendix D lists a conversion program for the IBM PC which generates TEKHEX source files from the Y binary image files; the monitor-oriented downloader is described below.

A Tektronix 8550 Microprocessor Development system (MDS) used at NASA Langley Research Center in conjunction with their DATAC system research. The TekHex code format generated by the MDS requires checksum calculation and handshaking with the host system downloader. This method of program loading is useful with a host that can support bidirectional communication between itself and the Bus Monitor and is intended for use with high speed serial lines.
Figure 3.10. Z8001 CPU Programmer's Model
Figure 3.11. Y Assembler Binary File Format
The TekHex file transfer protocol and format are described in Appendix E.

To facilitate program downloading from the MDS to the Z8002 system when the MDS was not being used in emulator mode, an assembly language program was prepared for the S100 bus system. This program provided the necessary host prompts and ASCII-to-hexadecimal conversion necessary to receive the data stream from the host computer, determine the integrity of the message, reconstitute and store the program information. The Z8KLOADER program for the Z8002 is given in Appendix F.

The TekHex loader originally prepared to be used with the 8550 MDS can be used with any host that provides the TekHex protocol. Because of limited access to the MDS and the convenience of using a microcomputer as a Bus Monitor Console system, a host version of the downloader was written in the C language for the CP/M operating system.

Originally, the console system was originally a Digital Equipment Corporation (DEC) VT-180 system. Failure of the VT-180 disk drive system late in the program, the emergence of the IBM PC architecture as the de facto standard in small-scale computing systems, and the overall greater flexibility of the PC family have caused the PC to displace the VT-180. It was this conversion that prompted the development of the second format. A non-system specific version of the host downloader is presented in Appendix G.

A simpler approach was used to provide a fast means of preparing a program downloader to run on the PC. The CP/M version of
the TekHex downloader used machine language functions [24, 25] to ensure that the VT-180 did not drop bytes sent serially from the Z8002. When the PC was installed, rather than attempt to learn the inner workings of MS/PC-DOS to recreate the patches (a task complicated by the fact that MS-DOS has a known bug in its serial device handler which requires a patch to work correctly [26]), a simpler approach to program downloading was dictated to recover time lost due to the demise of the VT-180. This method has proven adequate for the laboratory environment, although no tests of its performance in a high noise environment have been performed.

This second ASCII format takes an open-loop approach to program downloading. The Trump Card binary file is scanned and reconstituted into the ASCII characters corresponding to the machine code. The MPU8000 II monitor's "Examine" ("E") command is sent to a disk file along with the starting address(es) of the data and then the data itself. The result is an ASCII file containing this data with the extension .ZHX to suggest Z8002 HeX file. The file is then sent a byte at a time to the console port of the Bus Monitor system. The MPU8000 II processes the data stream as though it were a series of very rapid memory-deposit commands. Other MPU8000 II monitor commands and their formats are presented in Appendix H.

Figure 3.12 illustrates a sample of this format of the data transfer. After the "E" command is entered with an address argument, that address and the current word value are displayed. A new value is entered on the same line, followed by a carriage return (0D hexadecima)l. The next address and its datum are then displayed and
Figure 3.12. "ZHX" File Format for Program Downloading to Bus Monitor
modified. This process is repeated until an ESC character (1B hex) is entered to terminate the memory-deposit operation. Appendix I describes the program which translates the Y binary file to this ASCII format.

2. Basic Z8002 Software Features. A DATAC message sent from the terminal currently consists of up to 2048 16-bit words reconstituted by the DATAC terminal. While the 16-bit address space allows for up to 32K words, the current SIR can only accommodate 2k words. The message's information content is determined by two items, a label used exclusively by the bus terminal which indirectly specifies the source of the message, and the memory address where the DATAC terminal will store the data in the common memory shared by the terminal and the subsystem it serves. Ordinarily, a given subsystem will receive only one kind of data and will need only an address to determine the nature of the data deposited in its memory by its bus terminal. However, the Bus Monitor must deal with having to receive all data appearing on the bus. It will have to process both the message label to identify the transmission source and then use the datum address to determine what kind of information is being stored there so that the datum may be processed later.

Fortunately, several factors simplify the data identification problem. One is that the subsystem interface of each DATAC terminal supports a 16-bit address space of 64K bytes or 32k words. The bus terminal for the Bus Monitor can be programmed to locate the messages at whatever address is convenient for the subsystem. In the case of the Bus Monitor, structures like exception tables and program code
and data must be protected from being overwritten by an incoming message. Another factor is that the current ATOPS vehicle operates with a message traffic volume of approximately 1500 words. Given the 32k address space available from the bus terminals, there is adequate room to locate messages sent to the Bus Monitor.

Since each datum is assigned a unique SIR address, there is no constraint to locate data at an address which might encumber Bus Monitor programs. Since a Major-Frame message block will fit into 3k words, and given the memory space available from the S100 system, the Bus Monitor is capable of storing several frames of messages simultaneously.

Thus the Bus Monitor may freely locate the messages in memory so that a simple offset into the message block may be used to locate data. These factors when combined allow the Bus Monitor to organize message data in the most efficient and convenient manner. The software conducting the post processing will then use the message ID tag to provide a template for each block of data. In this manner, the burden of interpreting data is shifted to the PC, which has more processing time.

After some vector and table initialization, the Z8002 idles, waiting for the first message block from the DATAC terminal. When this message appears, the terminal interrupts the Z8002 and deposits the transmitter ID tag into the tag latch. The Z8002 reads the latch, saves the tag value, and, depending on the mode of operation, elects to process or ignore the transmission. If the system intends to process the data, the ID tag is used to determine the address of
the last word of the transmission. The word is tested until it changes from its assigned null value (null depending upon the context of the data). At this time the Z8002 assumes the transmission is concluded and optionally takes steps to move the data from the shared RAM area into memory inaccessible by the bus terminal. This is possible because only 12 of the available 16 address lines are brought out from the SSI on the DATAC terminal to the BlueBoard interface. The reason for moving the message is to prevent subsequent transmissions from corrupting the integrity of the message. Figure 3.13 illustrates these steps in flowchart form.

The Z8002's next action is to disable the DMA mechanism of the S100/Z-BUS interface. If the message-save operation is to be performed, then a block-move is initiated that moves the message into a holding area. Once all data have been moved out of the common area, a null value is placed at the last address referenced under that label.

Whether the message has been relocated or not, it is sent via an RS-232 port a byte at a time to the IBM PC. Once the message has been transferred, the DMA system of the BlueBoard is reenabled and the Z8002 waits for the next new message to process.

3. IBM PC Data Handler Routines. Depending on the mode of operation desired, the data processing programs for the IBM PC console system will vary greatly. The one common element between all such programs will be the need to accept serial data from the S100 system. In an operation quite similar to the Z8002 program to stream data from the bus terminal to the console device, the PC program will
Figure 3.13. Basic Bus Monitor Message Handling Routine
wait for an incoming serial message prefaced by a message identifier. This identifier is supplied by the Z8002 system prior to transmission to the PC and identifies the message to the PC in the same way that the interrupt vector/ID tag supplied by the bus terminal identified the message source to the S100 system. The principal (albeit minor) difference is that the PC will use the ID tag to look up a word count for the incoming message, whereas the Z8002 uses its ID tag to look up an address.

Once the PC has identified and received the message, it may process the data or store it for post-processing and then collect more data from the Z8002.

4. BlueBoard Interface for Z-BUS/S100 Signal Interchange. Since the S100 bus and Z-BUS are not directly compatible, an interface to connect the two systems is necessary. This interface has become known colloquially as the BlueBoard, so named because of the color of the prototyping board on which it was tested and built. To understand the functions of the BlueBoard, it is necessary to first describe the hardware and signal conventions of the two systems it will be connecting. In accordance with IEEE practice adopted with the publication of the IEEE-696 microcomputer bus standard, the active-low state or logical negation of a signal will be designated by a suffixed asterisk ("\*").

There are two features of the interface card that do not provide for terminal control of the Bus Monitor memory. These are: 1) the Z8002 address line extensions and 2) interrupt vector capture to be used for transmitter ID processing.
Acquisition of the interrupt vector capture posed another problem. Originally, it was planned to read this word as part of the data received from the DATA terminal, as the Z8002 interrupt processing architecture captures the interrupt vector number on its stack [27]. However, the interrupt vector appears on the steady address/vectored interrupt lines, not the address/data lines used for normal Z-BUS transactions. Further, the vector and the Interrupt Strobe (IS*) signal which accompanies the vector signal appears approximately 20 µs before the terminal is presenting data and ready to access the memory of the Z8002. A 16-bit latch located in the I/O space of the Z8002 processor provided a convenient solution. The Blueboard DMA and address extension subsystems are shown schematically in figure 3.14.

a. A8-A15 Address Line Extension. The MPU8000 II CPU card was originally intended to be part of an business/office computer system. As such, it was usually sold with a companion Memory Management Unit. This device used the upper eight bits of the Z8002's 16-bit address bus and mapped it to a logical address space of several megabytes. Since no need for such addressing capability was anticipated, the CPU was ordered without the MMU. Unfortunately, the upper byte of the address bus was used by the MMU and those address bits were not brought out to the CPU's S100 bus connector, effectively leaving a 16-bit microprocessor with an eight-bit address bus.

Refer to figure 3.14 for the address line extension circuit. The missing upper half of the address bus was supplied by using the
same signals provided to the MMU and generating those address bits explicitly. The Z8000 family uses a multiplexed address/data bus to conserve on the number of pins per DIP package. Two status signals, Address Strobe (AS*) and Data Strobe (DS*) are asserted at the appropriate time to indicate the type of information on the shared bus. The upper byte of the address bus is obtained from the TBUS [28] connector used by the Memory Manager to access the CPU's control lines. When the AS* is asserted, the high-order address bits are saved in a 74LS373 octal D-type latch (U8) which uses AS* as its clock. The latch outputs are then sent to U7, a 74LS244 three-state buffer. This buffer is controlled by the IEEE 696 ADSB* line (Address buffer DiSaBle*), so that the upper half of the address bus is disabled during the terminal access along with the lower half of the bus. This restores the Z8000 address bus to its nominal 64k limit.

b. **S100/Z-BUS DMA Transfer.** To move data from the DATAC terminal to the S100-based system, an interface that translates the Z-BUS DMA signals to their S100 counterparts was prepared. The actual translation of signals between the two different systems was accomplished by first defining which S100 features would be supported by the interface. The only requirement for this device is to allow the Bus Monitor's bus terminal a means to perform a DMA access on the S100 bus memory system. The interface would have to assure an orderly transfer of the bus from the Bus Monitor processor to the DATAC terminal, allow the DATAC terminal to access the Bus Monitor RAM for a read or write operation, and then return the S100 bus to
the Z8002 CPU.

Z-BUS DMA Transfers: The Z-BUS bus transfer is illustrated in figure 3.15. After a BUSREQ* signal is received, it is synchronized to the first falling edge of the system clock. By the first clock cycle of the next machine cycle (about 3 clock cycles), the Z8000 places its status, control, and Address/Data lines in a high-impedance state and asserts BUSACK* to signal to the requesting device that the system bus is available for its use.

After the requesting device has released BUSREQ*, that signal is again synchronized with the first rising edge of the system clock. One clock cycle later, BUSACK* is deasserted and the CPU restores its various lines to their previous states as the processor reexecutes the instruction interrupted by the DMA request.

S100 DMA Transfers: While the somewhat mnemonic S100 signal names will be explained in context, please refer to Appendix J for a full description of the signal functions.

The S100 bus transfer is more complex than the Z-BUS transfer. Unlike the Zilog DMA operation, there are several control and status lines involved in the transfer from bus master to bus slave. Figure 3.16 shows the various control and signal lines involved in the bus-transfer process. The slave device initiates a bus request by asserting the HOLD* signal, analogous to the Z-BUS BUSREQ*. At the first rising edge of the system clock, the host processor asserts pHLDA (processor HoLD Acknowledge) and the transfer procedure begins. During the transfer of the bus, both the bus master and the requesting slave device will drive certain bus lines to minimize loss
Figure 3.15. ZBUS Bus Transfer Timing
WHERE:
XFER1 = ADSB* SDSB* DODSB*
XFER2 = CDSB*

Figure 3.16. S100/IEEE-696 Bus Transfer Timing
of data due to errant control signals.

On the first falling edge after pHLDA is asserted, ADSB*, SDSB* and DODSB* (Address buffer DiSaBle, Status buffer DiSaBle and Data-Out buffer DiSaBle, respectively) are asserted by the bus transfer circuitry. These control lines disable the address bus buffers, status line buffers and the data bus output buffers as their names might suggest. At this time, both the bus master and the slave device have access to bus control lines. Both devices will drive pHLDA to its logic TRUE state and pWR* (processor WRite), pSYNC (processor SYNChronize), pSTVAL* (processor STatus VALid) and pDBIN* (processor Data Bus IN) to their logic FALSE state. On the next rising edge, CDSB* (Control buffer DiSaBle) is asserted, the bus master's control lines are disabled, and the bus transfer is complete; the temporary master is free to access the bus and its devices.

After the slave device is finished, a complementary process to bus acquisition begins. The slave releases the HOLD* signal. On the first rising edge of the system clock, the bus transfer mechanism releases CDSB*. Once again, both permanent and temporary masters are driving the control lines with the same states as in the first transfer. On the subsequent falling edge of the clock cycle, ADSB*, DODSB* and SDSB* are released, restoring control to the permanent master. The return of the bus is completed by the permanent master releasing pHLDA. At this point, the bus master resumes the instruction it was executing when interrupted by the DMA request.

Operation of the BlueBoard DMA subsystem: Refer to figure 3.14.
The incoming BUSREQ* signal is controlled by U35, a 74LS251 multiplexer (mux). When enabled, the mux allows the BlueBoard to process the BUSREQ* signal and the DMA operation to occur. The BUSREQ* signal passed from the DATAC subsystem interface through U35 is input to a 54221 TTL one-shot (U17b) with a period of 310 ns. The one-shot output pulse, along with BUSREQ* inverted by a 54LS04 buffer, goes to a dual-input NOR gate. One-shots are used to stretch the ersatz S100 signal widths to provide a more comfortable input margin to some of the TTL devices used in the BlueBoard. The output of the NOR gate forms the S100 signal HOLD*. The one-shot and the NOR gate extend the BUSREQ* signal to conform to the S100 timing for the HOLD* line. When the HOLD* signal is received, the bus master responds with pHLDA. This signal is fed to a D-type latch (U3a), whose Q* output is then ORed with the HOLD* signal. This produces a signal that is internal to the interface, called XFER1. This signal drives the ADSB*, SDSB*, and DODSB* control lines and enables certain bus control line drivers for the interface. The noninverted output of this first latch is fed to a second D-type latch (U3b). The inverted output of U3b is ORed with the original DATAC BUSREQ* signal to produce two more internal signals, XFER2 and its complement XFER2*. XFER2 drives CDSB* as well as enabling the interface's status line drivers while XFER2* is used to produce the pSYNC status line, which signifies the beginning of a new bus cycle.

The Q output of U3a triggers a 54221 one-shot (U17a) with a period of 220 ns. This pulse is ORed with Q* of U3b to form the BUSACK* signal for the DATAC Z-BUS. This signal is also routed to
the interface's bus control line driver to produce the signal \( \text{pSTVAL} \).

Appendices J and K present respectively a complete summary of the various signals used on the S100 and Z-BUS buses.

The aforementioned circuit provides for transfer of the bus from the Z8000 processor to the DATAC terminal and generates the control signals used by the terminal to read from or write to the Bus Monitor Unit memory. Management of the actual address and data information occurs in a more straightforward manner.

Data information appears on the multiplexed Address/Data lines (\( \text{ADO} - \text{AD15} \)), while address information is available from both the Address/Data lines as well as the Vectored Interrupt/Address lines (\( \text{VI/ADO} - \text{VI/AD15} \)). Since address information from the Z8002 appears at an acceptable time for the S100 bus cycles, the high order address lines are brought in directly from the subsystem interface into two 54/74LS244s, designated U5 and U6, each a byte-wide three-state buffer. The output enable pin on each buffer is enabled by the transition state \( \text{XFER1}^* \) being true. The S100 bus requires that address information be steady during the valid portion of \( \text{pSTVAL} \) and \( \text{pSYNC} \). The contents are ignored otherwise.

The data information from the DATAC terminal appears in a slightly different manner. Since data information is available only from the multiplexed address/data bus, these lines lead into two eight-bit bus transceivers (U9 and U10). Each of these 54/74LS245s is enabled by the \( \text{DS}^* \) signal presented from the subsystem interface, indicating that valid data is present. The direction of the data transfer, inbound to the terminal or outbound to the subsystem
device, is determined by the state of the R/W* line. Thus, the
terminal is able to read and write data to and from the Z8002
system's memory. The ability to access the shared memory in the
read-mode is important so that certain Bus Monitor features, such as
the echo-check, can be implemented in the future.

c. **Interrupt Vector Capture and DMA Management.**

Earlier, it was described how the interrupt-handling capability of
the DATAC terminal is used to provide transmission source
identification. The difficulty in using this interrupt vector as an
identification tag lies in the fact that it is not directly
accessible by the attached subsystem as data, which is how the Bus
Monitor processor would treat it. Rather than manipulate the Z8002's
address bus to obtain the interrupt vector during part of an
exception handling process (which, as described earlier, would be an
awkward procedure), it was decided instead to capture the 16-bit word
and use the interrupt strobe as a non-vectored interrupt.

Because of the modest interrupt needs of this system, it was
hoped to make use of the Non Vectored Interrupt (NVI) line of the
Z8002 to alert the S100 system that a new message transfer was taking
place. Once again the obvious was not possible. Because the S100
bus does not support any kind of NVI aside from RESET and NMI [29],
the NVI line on the Z8002 CPU board was pulled high on the MPU8000 II
CPU card. It was not practical to bring the NVI line out to an
unused pin on the S100 bus [30] because the operation of that line
would have to be subject to the same bus transfer mechanism (DMA
operations) as the rest of the CPU card systems and there was no
realistic way to provide that capability.

Vectored interrupts were disqualified because the hardware overhead involved in supplying an IRQ vector to the S100 environment was considered excessive. In light of the simple interrupts requirements of the Bus Monitor system and the lack of simpler alternatives, it was decided instead to gate off the NMI line using a 74LS251 mux in exactly the same way that the incoming BUSREQ* signal would be gated off. Flow of those two signals is, in fact, controlled by the same flip-flop. The decoding hardware for the NMI mux was already in place, having been installed to manage the multiplexer for the BUSREQ* line. Thus it was a far simpler matter to turn the Non Maskable Interrupt into a Now Maskable interrupt. The ID tag capture and DMA management subsystems of the BlueBoard are illustrated in figure 3.17.

ID Tag Capture: When a label is received by the Bus Monitor terminal, the interrupt vector, which the Bus Monitor treats as an ID tag, appears on the steady address lines and the interrupt strobe IS* is asserted. The tag capture circuitry appears in figure 3.17. Two 74LS374 octal D-type latches accomplish the task of retaining the ID tag by using the inverted IS* signal. The IS* is then routed to the Z8002's NMI line.

If U35 and U34 are enabled to pass the IS*/BUSREQ* signals, the IS*/NMI signal alerts the Z8002 that a transmission is being performed. The Bus Monitor needs the ID tag to determine, based on the current mode of operation, whether to process this block of messages or to ignore it. The two eight-bit latches holding the
Figure 3.17. BlueBoard DNA Management and ID Tag Capture
Interrupt vector/ID tag are mapped into the 64k I/O space of the Z8002 at address 01530H (it is Intel/Zilog convention to identify hexadecimal quantities with a suffixed "h" or "H," unlike Motorola's prefixed "$") for the upper byte of the word and 01D30H for the lower byte. Because of the instability of 16-bit I/O operations using the MPU8000 II/6SIO configuration, it was necessary to resort to byte-oriented operations.

On receipt of the NMI, the Z8002 fetches the ID tag using successive INB instructions, collecting one half of the ID tag at a time. The Bus Monitor processor then evaluates the tag and determines whether to continue its current data processing operation and wait for the next message, or monitor the transfer, determine when the transaction is complete, then save and process the data received.

DMA Management: While some data processing can be done by the Z8002, due to the time constraints imposed by incoming transmissions and the volume of data, most of the high level data reduction is performed by the IBM PC. The data on the Z8002 system is sent via RS-232 channel to the PC. Both the PC and the Z8002 system serial ports are limited to a maximum baud rate of 9600 baud. At 104 µs per bit each word will take a minimum of 2.083 mS to send. This is nearly 5% of the 50 mS currently in use as the ATOPS major frame rate. Clearly the speed of the serial channels presents a bottleneck: data is received from the terminal faster than the Z8002 can send it to the PC for processing.

To deal with the data overrun problem, it must be recognized
that for this hardware implementation, there is simply not enough
time available to the Z8002 to process the data flow. This arises
primarily from the aforementioned slowness of the serial channels
and, to a lesser extent, from the access time lost due to the bus-
terminal DMA access of the Z8002 memory. Until a means is found to
improve both of these items, it must be realized that for some modes
of operation, there will be a less-than-continuous stream of data
coming from the bus. To make this situation more manageable, a
simple form of DMA control is implemented on the DATAC terminal's
access to the Z8002 memory.

When accessing the Bus Monitor memory, the DATAC terminal is
supposed to wait for the assertion of the BUSACK* signal, indicating
that the host system is willing to release the bus to the terminal.
There will be occasions when it is desired to disallow DMA access of
Bus Monitor memory by the DATAC terminal. The primary reason for
this is to avoid both the bus terminal and the Bus Monitor from
attempting to access the same memory location at the same time. The
DMA arbitration cycle will prevent any corruption of the word being
transferred, but that alone does not ensure that every block of data
received from the bus will be valid. Since a given message block
contains words which have identical temporal significance, mixing
data words from different messages corrupts the overall content of
the message: individual data items will still be valid, but the
message itself has lost value because the amalgam of words no longer
has a useful time relationship among its components.

For example, if the Z8000 is trying to move a block of data
from system A's message from frame X and the terminal begins to
deposit data from system A from frame Y, the result would be a
mixture between the two message blocks. Because the identity of a
word in a message is dependent upon its location in a message,
messages from a given system always use the same SSI addresses.
Mixing the two messages would result in erroneous data
interpretation, particularly if a time-dependent analysis is
underway. An example would be monitoring the rate of change of a
given data item. A solution to this problem was to provide a means
to keep the terminal from accessing the Bus Monitor's memory when the
Bus Monitor needs to access the message in its memory. Ordinarily,
the Z-BUS protocol merely provides arbitration access time, so some
way to influence the DMA grant was necessary.

The DATAC terminal's ability to DMA the Z8002's memory is
dependent upon BUSREQ* being recognized and processed correctly. If
BUSREQ* is not received by the BlueBoard interface, the transfer
circuitry will never initiate the XFER1 and XFER2 cycles which 1)
force the Z8002 off the S100 bus and 2) enable the buffers that allow
the terminal access to the Bus Monitor memory. To this end, a DMA
switch has been implemented in the form of a JK flip-flop and a mux
located in the I/O space of the Z8002.

Refer to figure 3.17 for the hardware used to implement the DMA
switch. The I/O address 01130H is decoded to give a pulse on the J-
line and 01930H gives a pulse on the K-line. Thus any reference to
01130H will set the flip-flop; any reference to 01930H will clear it.
Since the address lines used to generate the J and K signals are
completely decoded, there is no chance for a toggle condition \((J=K=1)\) to occur. The output \(Q^*\) of the JK flip-flop leads to one input of each OR gate U38a and U38b. The other input of U38a is BUSREQ*; the output of the gate is routed to the HOLD* line on the S100 bus. Similarly, the second input to OR gate U38b is IS*. The output of U38b goes to the NMI* line of the Bus Monitor processor. When the JK latch is set \((Q^*\) is low) the output of U38a will follow the BUSREQ* line and a bus transfer is possible. When the latch is cleared \((Q^*\) is high), the HOLD* line will always be high and no transfer takes place. This same kind of isolation is provided on the IS* line, since the Z8002 should not be alerted to the presence of a potential transmission unless it is able to respond to it.

The buffers that manage data flow for the terminal side of the interface are enabled by XFER1*. This signal is initiated by the reception of the BUSREQ*/HOLD* signal. If the DMA process is not enabled, the terminal will have no access to the Z8002's memory and the assertion of the AS*, DS* and R/W* lines will have no effect. Thus, it is possible to manage the DMA of the Bus Monitor main processor memory to give priority to the Z8002's task as needed.

C. Summary

The BlueBoard interface provides full use of the 64k byte address space of the Z8002 microprocessor. It also allows a Z-BUS-compatible device to perform a DMA operation on an S100 bus memory device by requesting a bus transfer from the current bus master. In this implementation, the current bus master is the only processor on the bus: the Z8002 system. Provisions have been made to capture the
DATAC terminal 16-bit interrupt vector and make it accessible to the Z8002. This vector is treated as an ID tag so that the source of an incoming bus message may be determined. With access to the message data and the source of the transmission, qualitative and quantitative analysis of DATAC bus data transfers is possible.
IV. TEST PROCEDURE AND RESULTS

Tests were performed on the Bus Monitor Unit development system to determine how well it would be able to respond to the data stream from the simulated bus terminal subsystem interface. The simulation tests used Mode A rather than the more complex default mode of SCAT-C.

Because of the extensive software overhead in managing the control lines of the subsystem interface simulation, it was not possible to duplicate exactly the timing of the label and data words. Recall that in an actual DATAC system, a label word requires 24 μs for processing by the bus terminal; data words require 20 μs. The best simulation possible with the 68000 SBC was 24.5 μs for the data words and 30 μs for the labels. This translates to 122.5% and 125% of the actual data and label word times respectively. It will be shown that for this test procedure the timing differences are not particularly significant. The frame repetition rate of 50 ms was easily achieved and was used as a common reference point.

It can be argued that because SCAT-C is the optimal operating mode, it should be used to test the Bus Monitor system. Mode A was used for the tests for three reasons. First, given the limited processor power of the SBC used for the bus terminal SSI simulation, the Mode A simulation is simpler to program than SCAT-C and therefore is more likely to produce a temporally accurate simulation. Secondly, because the incoming Mode A data stream is simpler in structure, it is easier for the Bus Monitor Unit to interpret. Finally, it is easy to extrapolate Mode B/C results by adjusting the
50 ms results for the shorter 10 ms frames of the other operating modes.

A. Bus Monitor Unit Testing and Data Collection Procedures

The data stream consisted of 704 frames each containing 5 messages of equal length. That is, for a given frame, each message from the simulated bus subsystems contains the same number of words. The simulated bus system messages appear grouped at the beginning of the frame, followed by the bus quiet-time that is used for system expansion. 50 ms after the beginning of the current frame, a new frame simulation begins, sending similar messages with identical timing. Given the 32k byte memory of the SBC, there was room for only 3520 message descriptions once the terminal SSI simulation program had been loaded (704 frames of 5 systems sending messages gives 3520 messages simulated for each run of the experiment). While the 68000 CPU on the SBC has an address space of 24 MB, the fragmented memory map and the unbuffered system bus made RAM expansion very awkward. Fortunately, 704 frames produces a simulation run 35.2 seconds in duration. This is adequate to test BMU response to a continuous stream of data.

The word-count-per-message begins at 1 word and is increased to 99. Each word-count constitutes a trial, with each trial consisting of 10 runs. The data words in the messages are coded to reflect the "originating bus subsystem" and the current frame number. The 16-bit data words are coded thus: SFFF, where S is the four-bit digit bus station identification number and FFF is the 12-bit quantity indicating the frame number of the message that is composed of a
block of this and other identical words. For example, a data word of 407B would be decoded in this way: S = "4," so the "originating system" was system number 4; FFF = "07B," 07B in base 16 is decimal 123, so the data word was part of a message "sent" by system number 4 during frame 123. Messages were coded in this way so that it was possible to extract timing information about message reception based on the frame number. This eliminated the need for addition Bus Monitor software or hardware on the Z8002 system to provide this information.

Test software for the Z8002 and PC systems was very similar to the general description of the Bus Monitor programs described in Chapter 3. The Z8002 unit waits for a message from the bus terminal simulation. When a message transfer begins, the Z8002 collects the ID tag supplied by the VI/AD lines of the SSI and gets the address of the last word to be deposited under that ID. Once that word has changed from a predefined null value, the Z8002 disables the terminal access mechanism and serially sends the message to the PC 8 bits at a time via an RS-232 serial link. The PC side idles, waiting to collect data from the serial port attached to the Z8002 system. When data appears, the PC saves it in a 64K RAM buffer and checks the keyboard status. If the "ESC" key is pressed on the PC keyboard, the test run is concluded and the contents of the RAM buffer are written to a disk file for later analysis. A partial hex dump of a typical COMSAVE data file is shown in table 4.1. The TXMSG program for the Z8002 and the COMSAVE program for the PC are described in Appendices L and M, respectively.
### Table 4.1. Hexadecimal Data Collected By COMSAVE Program from S100 System

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>10 00 10 00 10 01 10 01 10 02 10 02 *</td>
</tr>
<tr>
<td>0010</td>
<td>10 02 10 03 10 03 10 04 10 04 10 05</td>
</tr>
<tr>
<td>0020</td>
<td>10 05 10 06 10 06 10 06 10 07 10 07 *</td>
</tr>
<tr>
<td>0030</td>
<td>10 08 10 08 10 08 10 09 10 09 10 0A</td>
</tr>
<tr>
<td>0040</td>
<td>10 0A 10 0B 10 0B 10 0B 10 0C 10 0C 10 0D</td>
</tr>
<tr>
<td>0050</td>
<td>10 0D 10 0E 10 0E 10 0F 10 0F 10 0F</td>
</tr>
<tr>
<td>0060</td>
<td>10 10 10 10 10 10 11 10 11 10 12 10 12 *</td>
</tr>
<tr>
<td>0070</td>
<td>10 12 10 13 10 13 10 14 10 14 10 14 10 15</td>
</tr>
<tr>
<td>0080</td>
<td>10 14 10 15 10 16 10 16 10 16 10 17 10 17</td>
</tr>
<tr>
<td>0090</td>
<td>10 18 10 18 10 19 10 19 10 19 10 1A</td>
</tr>
<tr>
<td>00A0</td>
<td>10 1A 10 1B 10 1B 10 1C 10 1C 10 1D</td>
</tr>
<tr>
<td>00B0</td>
<td>10 1D 10 1E 10 1E 10 1E 10 1F 10 1F</td>
</tr>
<tr>
<td>00C0</td>
<td>10 20 10 20 10 20 10 21 10 21 10 21</td>
</tr>
<tr>
<td>00D0</td>
<td>10 22 10 23 10 23 10 24 10 24 10 25</td>
</tr>
<tr>
<td>00E0</td>
<td>10 25 10 26 10 26 10 26 20 27 20 27</td>
</tr>
<tr>
<td>00F0</td>
<td>10 28 10 29 10 29 10 29 10 29 10 2A</td>
</tr>
<tr>
<td>0100</td>
<td>10 2A 10 2B 10 2B 10 2B 10 2B 10 2C</td>
</tr>
<tr>
<td>0110</td>
<td>10 2D 10 2E 10 2E 10 2E 10 2F 10 2F</td>
</tr>
<tr>
<td>0120</td>
<td>10 30 10 30 10 31 10 31 10 31 10 32</td>
</tr>
<tr>
<td>0130</td>
<td>10 32 10 33 10 33 10 34 10 34 10 34 10 35</td>
</tr>
<tr>
<td>0140</td>
<td>10 35 10 36 10 36 10 36 10 37 10 37</td>
</tr>
<tr>
<td>0150</td>
<td>10 38 10 39 10 39 10 39 10 39 10 3A</td>
</tr>
<tr>
<td>0160</td>
<td>10 3A 10 3B 10 3B 10 3B 10 3C 10 3C</td>
</tr>
<tr>
<td>0170</td>
<td>10 3D 10 3E 10 3E 10 3E 10 3F 10 3F</td>
</tr>
<tr>
<td>0180</td>
<td>10 40 10 40 10 40 10 41 10 41 10 41</td>
</tr>
<tr>
<td>0190</td>
<td>10 42 10 43 10 43 10 44 10 44 10 44</td>
</tr>
<tr>
<td>01A0</td>
<td>10 45 10 45 10 46 10 46 10 46 10 47</td>
</tr>
</tbody>
</table>
B. Test Results and Assessment of COMSAVE Data

Each data file collected by the COMSAVE program was first formatted by DSCAN. The purpose of this program is to tabulate the message data present in the file and display the ID number of the bus subsystem which was the source of the message, the frame number in which the message appeared, and the number of data words in those messages. A sample of a DSCAN output file is shown in table 4.2; the DSCAN program listing is given in Appendix N.

Each tabulated data file was imported into a template prepared for the Lotus 1-2-3 spreadsheet program. This template accepts DSCAN data files and provide the percentage of bus system messages processed, the average message frame adjacency (the number of frames separating successive messages processed by the Z8002 system) and the average data bandwidth.

The Percent Messages Processed was obtained by dividing the number of messages received by 3520 total messages. The number of messages received was obtained by counting the number of unique entries in each DSCAN file using the 1-2-3 @COUNT function [31].

Adjacency was determined by subtracting a given frame number from its successor. For example, the adjacency between frames 213 and 220 would be 7. Each entry from a DSCAN file provided a frame adjacency number, all of which were averaged using the @AVG function [32]. This provided the Average Frame Adjacency for each run.

Average Data Bandwidth is the reciprocal of the message availability time. Each simulated frame occurs at 50 ms intervals. This translates to a data bandwidth of 20 Hz for any given data word.
Table 4.2. Sample Data Output by DSCAN Program

<table>
<thead>
<tr>
<th>Station Number</th>
<th>Frame Number</th>
<th>Number of Words Received</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0000</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>0001</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>0002</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>0003</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>0004</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>0005</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>0006</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>0007</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>0008</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>0009</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>0010</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>0011</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>0012</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>0013</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>0014</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>0015</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>0016</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>0017</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>0018</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>0019</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>0020</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>0021</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>0022</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>0023</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>0024</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>0025</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>0026</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>0027</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>0028</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>0029</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>0030</td>
<td>3</td>
</tr>
</tbody>
</table>
When the Bus Monitor begins to skip frames, the bandwidth decreases: if one frame is skipped, the Bus Monitor is processing one-half of the frames and is providing a new data word or words only once every 100 ms or at a 10 Hz rate. As the frame adjacency increases, more frames are skipped and the data bandwidth of the Bus Monitor decreases accordingly. The Average Data Bandwidth is defined as the reciprocal of the frame adjacency multiplied by the 50 ms frame repetition rate.

For every trial, these three statistics were collected for each of 10 runs and then averaged to produce one set of points for that trial. The 99 points were plotted as a function of the words-per-message. The Percentage Messages Processed, Average Frame Adjacency, and Average Data Bandwidth are presented as figures 4.1, 4.2 and 4.3, respectively. The data collected are presented in Table 4.3.

Each graph is a piecewise-continuous linear function. The discontinuities occur at points where more than \( n \)50 ms are required for serial transmission of the current bus system message from the Z8002 system to the PC, where \( n \) is a positive non-zero integer. As \( n \) increases, the data in subsequent messages (and, as \( n \) increases further, data in subsequent frames) will be ignored as the Z8002 spends more time sending the current message via RS232. These discontinuities illustrate points where the frame adjacency abruptly changes, which means that another frame is being skipped. It can be predicted that for the decreasing functions, the values at the discontinuities will be 1/2, 1/3, 1/4 and 1/5 of the best case values as 1, 2, 3 and 4 frames are skipped while serially transmitting the
Figure 4.1. Percentage of Messages Processed By Bus Monitor as a Function of Message Word Count
Figure 4.2. Average Frame Adjacency of Bus Messages as a Function of Message Word Count
Figure 4.3. Average Data Bandwidth of Bus Monitor as a Function of Message Word Count.
Table 4.3. Summary of Data Collected by COMSAVE/TxMSG Programs

<table>
<thead>
<tr>
<th>number of words per message</th>
<th>number of messages processed</th>
<th>% messages processed</th>
<th>average frame adjacency</th>
<th>average data bandwidth</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 749.1 21.28124</td>
<td>0.938491 21.31151</td>
<td>0.938491 21.31151</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 703.9 19.99715</td>
<td>0.998721 20.02559</td>
<td>0.998721 20.02559</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3 703.8 19.99431</td>
<td>0.998663 20.02275</td>
<td>0.998663 20.02275</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 703.9 19.99715</td>
<td>0.998721 20.02559</td>
<td>0.998721 20.02559</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5 703.8 19.99431</td>
<td>0.998663 20.02275</td>
<td>0.998663 20.02275</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6 703.8 19.99431</td>
<td>0.998663 20.02275</td>
<td>0.998663 20.02275</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7 703.5 19.98579</td>
<td>0.999289 20.01422</td>
<td>0.999289 20.01422</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8 703.7 19.99147</td>
<td>0.999005 20.0199</td>
<td>0.999005 20.0199</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9 703.8 19.99431</td>
<td>0.998663 20.02275</td>
<td>0.998663 20.02275</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10 703.5 19.98579</td>
<td>0.999289 20.01422</td>
<td>0.999289 20.01422</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11 703.5 19.98579</td>
<td>0.999289 20.01422</td>
<td>0.999289 20.01422</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12 703.6 19.98863</td>
<td>0.999147 20.01706</td>
<td>0.999147 20.01706</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13 703.7 19.99147</td>
<td>0.999005 20.0199</td>
<td>0.999005 20.0199</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14 703.1 19.97443</td>
<td>0.999857 20.00284</td>
<td>0.999857 20.00284</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15 703.5 19.98579</td>
<td>0.999289 20.01422</td>
<td>0.999289 20.01422</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16 703.5 19.98579</td>
<td>0.999289 20.01422</td>
<td>0.999289 20.01422</td>
<td></td>
<td></td>
</tr>
<tr>
<td>17 703.2 19.97727</td>
<td>0.999715 20.00566</td>
<td>0.999715 20.00566</td>
<td></td>
<td></td>
</tr>
<tr>
<td>18 703.4 19.98295</td>
<td>0.999431 20.01137</td>
<td>0.999431 20.01137</td>
<td></td>
<td></td>
</tr>
<tr>
<td>19 703.3 19.98011</td>
<td>0.999573 20.00853</td>
<td>0.999573 20.00853</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20 703.2 19.97727</td>
<td>0.999715 20.00566</td>
<td>0.999715 20.00566</td>
<td></td>
<td></td>
</tr>
<tr>
<td>21 703.2857 19.9797</td>
<td>0.999594 20.00812</td>
<td>0.999594 20.00812</td>
<td></td>
<td></td>
</tr>
<tr>
<td>22 644.8571 18.3198</td>
<td>1.090165 18.34585</td>
<td>1.090165 18.34585</td>
<td></td>
<td></td>
</tr>
<tr>
<td>23 468.8571 13.3198</td>
<td>1.497564 13.35502</td>
<td>1.497564 13.35502</td>
<td></td>
<td></td>
</tr>
<tr>
<td>24 351.8571 9.995941</td>
<td>1.995129 10.02442</td>
<td>1.995129 10.02442</td>
<td></td>
<td></td>
</tr>
<tr>
<td>25 352 10</td>
<td>1.994318 10.02849</td>
<td>1.994318 10.02849</td>
<td></td>
<td></td>
</tr>
<tr>
<td>26 352 10</td>
<td>1.994318 10.02849</td>
<td>1.994318 10.02849</td>
<td></td>
<td></td>
</tr>
<tr>
<td>27 352 10</td>
<td>1.994318 10.02849</td>
<td>1.994318 10.02849</td>
<td></td>
<td></td>
</tr>
<tr>
<td>28 352 10</td>
<td>1.994318 10.02849</td>
<td>1.994318 10.02849</td>
<td></td>
<td></td>
</tr>
<tr>
<td>29 352 10</td>
<td>1.994318 10.02849</td>
<td>1.994318 10.02849</td>
<td></td>
<td></td>
</tr>
<tr>
<td>30 352 10</td>
<td>1.995535 10.02237</td>
<td>1.995535 10.02237</td>
<td></td>
<td></td>
</tr>
<tr>
<td>31 351.8571 9.995941</td>
<td>1.996348 10.01831</td>
<td>1.996348 10.01831</td>
<td></td>
<td></td>
</tr>
<tr>
<td>32 352 10</td>
<td>1.994318 10.02849</td>
<td>1.994318 10.02849</td>
<td></td>
<td></td>
</tr>
<tr>
<td>33 352 10</td>
<td>1.995129 10.02441</td>
<td>1.995129 10.02441</td>
<td></td>
<td></td>
</tr>
<tr>
<td>34 352 10</td>
<td>1.995535 10.02237</td>
<td>1.995535 10.02237</td>
<td></td>
<td></td>
</tr>
<tr>
<td>35 352 10</td>
<td>1.996347 10.01829</td>
<td>1.996347 10.01829</td>
<td></td>
<td></td>
</tr>
<tr>
<td>36 352 10</td>
<td>1.995535 10.02237</td>
<td>1.995535 10.02237</td>
<td></td>
<td></td>
</tr>
<tr>
<td>37 352 10</td>
<td>1.995941 10.02033</td>
<td>1.995941 10.02033</td>
<td></td>
<td></td>
</tr>
<tr>
<td>38 352 10</td>
<td>1.995129 10.02441</td>
<td>1.995129 10.02441</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>-----</td>
<td>------</td>
<td>---------</td>
<td>---------</td>
</tr>
<tr>
<td>39</td>
<td>352</td>
<td>10</td>
<td>1.995941</td>
<td>10.02033</td>
</tr>
<tr>
<td>40</td>
<td>351.7142</td>
<td>9.991882</td>
<td>1.997159</td>
<td>10.01423</td>
</tr>
<tr>
<td>41</td>
<td>351.4285</td>
<td>9.983765</td>
<td>1.999194</td>
<td>10.00406</td>
</tr>
<tr>
<td>42</td>
<td>352</td>
<td>10</td>
<td>1.995129</td>
<td>10.02441</td>
</tr>
<tr>
<td>43</td>
<td>351.7142</td>
<td>9.991882</td>
<td>1.997971</td>
<td>10.01016</td>
</tr>
<tr>
<td>45</td>
<td>291.1428</td>
<td>8.271103</td>
<td>2.375172</td>
<td>8.420734</td>
</tr>
<tr>
<td>46</td>
<td>275.2857</td>
<td>7.820616</td>
<td>2.550173</td>
<td>7.842907</td>
</tr>
<tr>
<td>47</td>
<td>235.5714</td>
<td>6.692369</td>
<td>2.978787</td>
<td>6.714174</td>
</tr>
<tr>
<td>48</td>
<td>235</td>
<td>6.676136</td>
<td>2.987234</td>
<td>6.695156</td>
</tr>
<tr>
<td>49</td>
<td>235</td>
<td>6.676136</td>
<td>2.986018</td>
<td>6.697884</td>
</tr>
<tr>
<td>50</td>
<td>235</td>
<td>6.676136</td>
<td>2.987234</td>
<td>6.695156</td>
</tr>
<tr>
<td>51</td>
<td>235</td>
<td>6.676136</td>
<td>2.987234</td>
<td>6.695156</td>
</tr>
<tr>
<td>52</td>
<td>235</td>
<td>6.676136</td>
<td>2.987841</td>
<td>6.693795</td>
</tr>
<tr>
<td>53</td>
<td>235</td>
<td>6.676136</td>
<td>2.987841</td>
<td>6.693795</td>
</tr>
<tr>
<td>54</td>
<td>235</td>
<td>6.676136</td>
<td>2.987841</td>
<td>6.693795</td>
</tr>
<tr>
<td>55</td>
<td>235</td>
<td>6.676136</td>
<td>2.989665</td>
<td>6.689714</td>
</tr>
<tr>
<td>56</td>
<td>235</td>
<td>6.676136</td>
<td>2.989057</td>
<td>6.691074</td>
</tr>
<tr>
<td>57</td>
<td>235</td>
<td>6.676136</td>
<td>2.989057</td>
<td>6.691074</td>
</tr>
<tr>
<td>58</td>
<td>235</td>
<td>6.676136</td>
<td>2.988449</td>
<td>6.692435</td>
</tr>
<tr>
<td>59</td>
<td>235</td>
<td>6.676136</td>
<td>2.989057</td>
<td>6.691074</td>
</tr>
<tr>
<td>60</td>
<td>235</td>
<td>6.676136</td>
<td>2.990273</td>
<td>6.688353</td>
</tr>
<tr>
<td>61</td>
<td>235</td>
<td>6.676136</td>
<td>2.989057</td>
<td>6.691074</td>
</tr>
<tr>
<td>62</td>
<td>235</td>
<td>6.676136</td>
<td>2.989665</td>
<td>6.689714</td>
</tr>
<tr>
<td>63</td>
<td>235</td>
<td>6.676136</td>
<td>2.989665</td>
<td>6.689714</td>
</tr>
<tr>
<td>64</td>
<td>235</td>
<td>6.676136</td>
<td>2.990273</td>
<td>6.688353</td>
</tr>
<tr>
<td>65</td>
<td>235</td>
<td>6.676136</td>
<td>2.989057</td>
<td>6.691074</td>
</tr>
<tr>
<td>66</td>
<td>220.1428</td>
<td>6.254058</td>
<td>3.186895</td>
<td>6.275705</td>
</tr>
<tr>
<td>67</td>
<td>211</td>
<td>5.994318</td>
<td>3.322274</td>
<td>6.019971</td>
</tr>
<tr>
<td>68</td>
<td>201</td>
<td>5.710227</td>
<td>3.482587</td>
<td>5.742857</td>
</tr>
<tr>
<td>69</td>
<td>201</td>
<td>5.710227</td>
<td>3.491115</td>
<td>5.728832</td>
</tr>
<tr>
<td>70</td>
<td>176.1428</td>
<td>5.004058</td>
<td>3.976483</td>
<td>5.029569</td>
</tr>
<tr>
<td>71</td>
<td>176.1428</td>
<td>5.004058</td>
<td>3.976483</td>
<td>5.029569</td>
</tr>
<tr>
<td>72</td>
<td>176</td>
<td>5</td>
<td>3.977272</td>
<td>5.028571</td>
</tr>
<tr>
<td>73</td>
<td>176</td>
<td>5</td>
<td>3.977272</td>
<td>5.028571</td>
</tr>
<tr>
<td>74</td>
<td>176</td>
<td>5</td>
<td>3.977272</td>
<td>5.028571</td>
</tr>
<tr>
<td>75</td>
<td>176</td>
<td>5</td>
<td>3.977272</td>
<td>5.028571</td>
</tr>
<tr>
<td>76</td>
<td>176</td>
<td>5</td>
<td>3.978083</td>
<td>5.027546</td>
</tr>
<tr>
<td>77</td>
<td>176</td>
<td>5</td>
<td>3.978083</td>
<td>5.027546</td>
</tr>
<tr>
<td>78</td>
<td>176</td>
<td>5</td>
<td>3.978083</td>
<td>5.027546</td>
</tr>
<tr>
<td>79</td>
<td>176</td>
<td>5</td>
<td>3.979707</td>
<td>5.025496</td>
</tr>
<tr>
<td>80</td>
<td>176</td>
<td>5</td>
<td>3.977272</td>
<td>5.028571</td>
</tr>
<tr>
<td>81</td>
<td>176</td>
<td>5</td>
<td>3.978895</td>
<td>5.026521</td>
</tr>
<tr>
<td>82</td>
<td>176</td>
<td>5</td>
<td>3.978895</td>
<td>5.026521</td>
</tr>
<tr>
<td>83</td>
<td>176</td>
<td>5</td>
<td>3.978083</td>
<td>5.027546</td>
</tr>
<tr>
<td>84</td>
<td>176</td>
<td>5</td>
<td>3.978895</td>
<td>5.026521</td>
</tr>
<tr>
<td>85</td>
<td>176</td>
<td>5</td>
<td>3.979707</td>
<td>5.025496</td>
</tr>
<tr>
<td>86</td>
<td>176</td>
<td>5</td>
<td>3.982954</td>
<td>5.021403</td>
</tr>
<tr>
<td>87</td>
<td>176</td>
<td>5</td>
<td>3.987824</td>
<td>5.015269</td>
</tr>
</tbody>
</table>
Table 4.3. Summary of Data (continued)

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>88</td>
<td>167.1428</td>
<td>4.748376</td>
<td>4.188041</td>
<td>4.775506</td>
<td></td>
</tr>
<tr>
<td>89</td>
<td>162.2857</td>
<td>4.610388</td>
<td>4.318677</td>
<td>4.631059</td>
<td></td>
</tr>
<tr>
<td>90</td>
<td>156.8571</td>
<td>4.456168</td>
<td>4.470852</td>
<td>4.473418</td>
<td></td>
</tr>
<tr>
<td>91</td>
<td>155</td>
<td>4.403409</td>
<td>4.523502</td>
<td>4.421363</td>
<td></td>
</tr>
<tr>
<td>92</td>
<td>141</td>
<td>4.005681</td>
<td>4.964539</td>
<td>4.028571</td>
<td></td>
</tr>
<tr>
<td>93</td>
<td>141</td>
<td>4.005681</td>
<td>4.964539</td>
<td>4.028571</td>
<td></td>
</tr>
<tr>
<td>94</td>
<td>141</td>
<td>4.005681</td>
<td>4.961499</td>
<td>4.03104</td>
<td></td>
</tr>
<tr>
<td>95</td>
<td>141</td>
<td>4.005681</td>
<td>4.963525</td>
<td>4.029394</td>
<td></td>
</tr>
<tr>
<td>96</td>
<td>141</td>
<td>4.005681</td>
<td>4.963525</td>
<td>4.029394</td>
<td></td>
</tr>
<tr>
<td>97</td>
<td>141</td>
<td>4.005681</td>
<td>4.964539</td>
<td>4.028571</td>
<td></td>
</tr>
<tr>
<td>98</td>
<td>141</td>
<td>4.005681</td>
<td>4.964539</td>
<td>4.028571</td>
<td></td>
</tr>
<tr>
<td>99</td>
<td>141</td>
<td>4.005681</td>
<td>4.964539</td>
<td>4.028571</td>
<td></td>
</tr>
</tbody>
</table>
current message.

For example, in figure 4.1, the percentage of messages processed decreases from 20% to approximately 7%, then to 6, 5, and finally about 4%. These figures result from the Z8002 ignoring messages from systems 2 through 5 while transmitting the message from system 1 and thus is sending only 1/5 (20%) of the data that the bus terminal is trying to send it.

When the frame adjacency rises to 2, only one message out of two complete frames (10 messages total) is being processed and so the percentage drops to 10%. When adjacency is 3, 1 message out of 15 is being sent to the PC, and so on. When adjacency rises to 5, the Bus Monitor is processing only 4% of the messages destined for it.

A more useful interpretation of adjacency is the Average Data Bandwidth (fig. 4.3). At the first inflection point, the data update rate falls from 20 Hz to 10. At successive discontinuities the data rate falls to about 6.67 Hz, to 5 Hz, and finally to 4 Hz with an adjacency of 5. At this point, the Bus Monitor is only able to update data once every 250 ms., which is slower than the minimum update rate specified by the current commercial avionics data bus, ARINC 429. The 429 specification calls for the most critical data to be updated every 20 ms. or at 50 Hz [33]; the least critical at 5 Hz or every 200 ms. Attempting to monitor more than 88 words will prevent the Bus Monitor from processing even the slowest system parameter within its specified time interval.

As an aside, if the Bus Monitor programs for the Z8002 provided for the S100 system to wait until all messages had been received
before sending data words to the PC, the initial percentages of messages processed would be higher. This would shift the inflection points to the left by a factor of 0.2. The overall characteristics of the frame adjacency and average bandwidth graphs would be relatively unchanged. However the abscissa of all the graphs would have to be "total words transmitted," rather than "number of words per message," due to the aggregate data transfer.

C. Validity of Bus Terminal Interface Simulation

Because of the overhead associated with managing the control lines of the bus terminal subsystem interface, it was not possible to prepare a simulation of the interface whose timing conforms exactly to that of an actual bus terminal. An actual bus terminal connected to a live DATAC system will present data words at the rate of 1 every 20 μs. Because of additional bits used to distinguish labels from data words, a label requires 24 μs of conversion by the bus terminal. Typical terminal gap values used in the bus access arbitration process are multiples of 1 microsecond, so the time between the last data word of one message and the label of the following message will be variable, but is reasonably close to the 30 μs obtained from the SSI simulation.

The bus terminal interface simulation allows for 24.5 μs per data word. This is 27.5% slower than actual word processing time. The magnitude of this difference suggests that the results obtained using the terminal interface simulation would not be applicable to real DATAC bus operation. Careful examination of the way in which the Bus Monitor operates will illustrate that these results are valid
within the range of message lengths that were used in the tests.

The basic operation performed by the Z8002 system in these tests is to allow data to be deposited into its memory, recognize the source of the data and send it serially to a PC for storage. A word deposit operation requires 20 µs; serial transfer of a word requires 2.083 ms. The difference between these two values is a factor of over 100 (actually 104.2). With a difference of 4.5 µs per word between the actual and simulated word deposit times, the simulated deposit of 88 words would take 396 µs longer than if the word/deposit occurred at the nominal 20 µs rate. However, it is not possible to send a word serially out of the S100 system in 396 µs. The point at which the simulated data stream slowness is noticeable occurs when the difference in word deposit times, multiplied by the number of words being serially transferred is an integral multiple of 2.083 ms. This means that the only time that the slowness of the simulated data stream becomes significant is when the added time for simulated word deposits takes away time that could be used to send words to the PC.

To determine this point, we set

\[ 2.083 \text{ ms} = n \times 4.5 \text{ µs}. \]

Solving for \( n \) we obtain \( n = 462.9 \). Thus, if the simulation transfers 463 words, the actual deposit operation would take \( \approx 2.083 \text{ ms} \) more than an actual terminal would require, effectively stealing time that could be used to send a word to the PC. Because it is the RS232 bottleneck that causes the system to cease to be useful in providing data real-time, the slowness of the word deposit operation is not significant for the volume of data used in these tests.
D. Applicability of Test Results to Alternate Modes of Operation

Because the SCAT-C is the default operating mode of the DATAC system, it is reasonable to question the use of Mode A tests in evaluating the Bus Monitor. The principal differences between the A and SCAT-C modes, at least as far as the Bus Monitor is or will be concerned, is that the message frames are shorter and that frame synchronization information is available under SCAT-C. Since this version of the Bus Monitor is not equipped to handle frame sync information, only the issue of frame duration remains.

We have identified that the time available to the Z8002 system to transmit fresh bus data to the waiting PC is the bus expansion time. The S100 system uses this time because there is no new data being received and so it can perform a data movement operation while minimizing the possibility of missing a new message.

At the 9600 bps rate of the S100 serial channel, word transmission time is fixed at 2.083 ms per word. The bus terminal word deposit time is such a small fraction of the serial transfer time that we may temporarily neglect it. The number of words that the Z8002 system can send to the PC in a given frame may then be approximated by dividing the expansion gap time of that frame by 2.083 ms/word, a simple linear expression. If the expansion gap time of a shorter frame can be expressed as a fraction of the expansion gap of Mode A, then it is reasonable to conclude that we may interpolate the S100 response to those shorter frames as being that same fraction of the Mode A results. Since the shortest frame
duration of the C or SCAT-C modes is 10 ms, the BMU response to data
arriving at that frame rate will be about 1/5 that of Mode A and its
50 ms frame rate.

Under Mode C, the subsystem messages are much shorter than in
Mode A. Because of the faster frame repetition rate, it is possible
to omit redundant information in a subsystem's message of one frame
and add it to the same subsystem's message for a subsequent frame.
Also, since the Mode C operations provide explicit bus timing
information, there is no need to keep message lengths identical from
frame to frame simply to provide subsystem timing coordination.

In each operating mode, the S100 system simply waits for data
words and then operates on them. Since the BMU currently has no
special tasks to perform contingent upon the bus operation mode, the
Mode C results are again likely to be 1/5 those of these Mode A
tests.

SUMMARY: Performance of this Bus Monitor is restricted by the
RS-232 serial link used to transfer data between the S100-based Z8002
data receiver/preprocessor and the MS-DOS PC used as the Bus Monitor
console device. Because of the large volume of data possible on the
DATAC bus and the rate at which it is transferred to the Bus Monitor,
the relative slowness of the serial ports restrict the flow out of
the Z8002 preprocessor system. This restriction requires that the
amount of data to be monitored be weighed against the need for having
the data of interest updated at the various rates provided by the
DATAC bus.

In terms of message update rates, as the number of frames
skipped increases, the bandwidth of the Bus Monitor decreases, reducing its ability to monitor aircraft parameters in real time. With no frames skipped, the Bus Monitor can process data at a 20 Hz rate, or a new datum every 50 ms. As the number of frames skipped rises, the bandwidth falls: with 1 frame skipped the bandwidth is 10 Hz, with 2 skipped 6.67, with 3 skipped 5 Hz, and with 4 frames skipped the bandwidth is reduced to 4 Hz. According to the specification for the ARINC 429 DITS Mark VII data distribution system, the slowest data update rate for an aircraft parameter is 5 Hz. Since DATAC is intended for commercial use (as is ARINC 429), we see that the Bus Monitor fails to provide useful updates while attempting to relay data from the bus at a rate greater than 87 words per 50 ms frame. If used in an alternate mode which utilizes the 10 ms frame update rate, the Bus Monitor performance decreases to processing no more than 17 words.

Thus, under Mode A, only 21 words can be monitored at the 20 Hz rate. This decreases in steps of 22 words to an acceptable minimum of 67 words at 5 Hz. Beyond that, information from the bus is being relayed slower than the slowest update rate specified by the ARINC 429 standard.
V. CONCLUSIONS

The Bus Monitor development system is capable of providing limited real time data from a DATAC bus. Mode A tests of the Bus Monitor show that it is capable of processing up to 22 data words per 50 ms frame without missing data from messages in the following frame. Attempting to monitor 22 to 44 words per frame will cause the Bus Monitor to ignore data presented in the following frame while it processes data from the preceding frame. Between 44 and 66 words the system will skip 2 frames between data updates. Three frames of messages are lost as the Bus Monitor processes between 66 and 88 words; processing 88 words or more causes the unit to miss 4 frames of data. Skipping more than 3 frames results in a data update rate of slower than 20 Hz, which is the slowest parameter update rate used by the ARINC 429 digital data bus currently in use with commercial avionics systems.

The inability to monitor all DATAC bus data in a real time frame is not necessarily a completely debilitating limitation. For developmental and diagnostic purposes, all that may be required is supervision of a word or two from a single system, or comparable data words from two or three separate systems. The current version of the Bus Monitor is able to provide the capability (with appropriate programming) of monitoring up to 22 systems, each providing one datum. The current ATOPS aircraft operates in flight mode with only five systems, so the data bandwidth limitation is not as severe as it might seem.

A second consideration is that not all data processing must be
done by the host/console unit, which relies on the serial link to obtain its data. Non-time-intensive tasks as bounds checking or pattern matching could, within limitations, be performed by the Z8002, avoiding the serial data bottleneck altogether. The ultimate criterion to determine the utility of this Bus Monitor is what kind of data analysis is required and whether the data to be analyzed comes from one or several different systems. For most laboratory needs, this Bus Monitor offers an acceptable degree of utility which offsets its primary limitation of data transfer speed.
VI. RECOMMENDATIONS

The current Bus Monitor configuration provides a means for limited streaming of data real time from the DATAC digital data bus. Data obtained from the bus terminal allows an operator to identify the source of a message that has appeared on the bus and monitor up to 22 words under a given label. Given this capability, it is desirable to be able to receive more data. It would be advantageous to process, for example, all words appearing under a label across successive frames or all words in a given Major or Minor frame. The ability to process large volumes of related data is useful in providing a comprehensive description of a subsystem and its bus activity. A logical extension of this Bus Monitor is to provide the capability to perform some kind of analysis or reduction of the data. For example, this could be reconstitution of hexadecimal data into real-world parameters, such as heading, airspeed, and so forth.

A. Limitations of the S100-based Bus Monitor

The current Bus Monitor does not have the capability to support the kinds of operations mentioned above. One restriction is that the Z8002 can directly address only 64K of memory. Compounding the limited memory space is the loss of 8k of RAM due to the common area of read/write memory used by the DATAC terminal to deposit data. This segment of memory cannot be used for program and data storage. The unavailability of this space, coupled with the loss of another 8k for the CPU card monitor ROM leaves 48k of RAM in which to develop code.

While the Z8000 family does have a very powerful instruction
set, it is quite easy to generate large amounts of code. Coupling large code size with the space consumed by a high-level language needed to perform data analysis results in big programs. Within that context, 48k of RAM becomes a limited resource.

Perhaps the most important consideration in providing these functions is the availability of sufficient time for the Z8002 to process the messages from the DATAC terminal. While the CPU runs at 6 Mhz, it must contend with relatively slow serial ports to send data to the console device for display and secondary processing. The Bus Monitor was constrained to use RS232 as the means for transferring data to the console because the DEC VT-180 originally intended to serve as the console had a closed architecture. The only way to communicate with the VT-180 was by way of its serial ports; implementation of a fast parallel transfer of data on that system was not possible.

With the replacement of the VT-180, the IBM PC and its open architecture have assumed the role of the console system. This has created many new opportunities for performing high-speed data transfers from the Z8002 system. Ideally, one would provide a channel between the S100 system and the PC which affords a data rate reasonably close to that of the 20 µs per word rate of the bus terminal.

B. Alternatives to Serial Data Transfer

One possibility is a direct DMA transfer between the S100 system and the PC. The SS-62 bus of the PC and its compatible systems provides DMA channels using a protocol conceptually similar
to the Z-BUS protocol [34]. The speed increase of using the DMA approach is marginally faster than a parallel-port arrangement and has the added advantage that the hardware to provide this function already exists in the PC system. Care must be taken to ensure that DMA requests by other PC subsystems, such as the dynamic RAM refresh or disk drive controllers are not affected. Also, SS-62 bus memory systems are byte-organized, not word-organized, which means that the hardware on the S100 system must be constrained to perform byte DMA transfers.

Another possibility is a simple parallel-port arrangement on both systems. A parallel data transfer between the S100 and IBM systems using readily available I/O devices has the potential of being almost an order of magnitude faster than the serial ports currently in use: 1.67 μs for a typical OUT instruction on the Z8002 as opposed to the 2.083 ms to send a 16-bit word via RS232. Dual octal latches (e.g., 'LS373) and hardware interrupts on the PC side would simplify handshaking and help speed the transfers. This simple change would alleviate most of the bottleneck the S100 system faces in processing the data from the DATAAC bus.

While adding a parallel port or DMA channel between the PC and the S100 systems seems simple and even obvious, one must consider the time and effort needed to support a system which is rapidly declining in terms of product viability, namely the S100 bus computer. Because the S100 bus was originally designed for the Intel family of eight-bit processors, its use in a 16 bit environment is awkward. This is exemplified by the use of separate unidirectional busses for byte
operations. The hardware needed to provide this kind of compatibility makes the S100 architecture unnecessarily complicated for Bus Monitor use. A schematic of the MPU8000 II CPU card appears in Appendix O.

In 1983 the S100 bus was a mature product with numerous vendors providing a wide variety of peripheral devices and support. In 1987, the IBM PC and PC/AT system architectures have emerged as the de facto standard in personal computing with an estimated installed base of more than nine million units [35]. A quick perusal of a small-systems computing magazine such as BYTE [36] attests to the rapid decline of the S100 system and the ubiquity of the IBM product (two advertisements for S100 systems versus literally hundreds for PC family products). At the very least it should be noted that the manufacturer of the MPU8000 II CPU and 6SIO serial cards is no longer operating and cannot provide any kind of service or support.

Rather than use an architecture with diminishing commercial support, the recommendation is made that the Bus Monitor be consolidated into the IBM PC. The Z8000-based system would be redesigned to act as a coprocessor with the 8088 of the PC, retaining its Z-BUS interface to simplify connection with the DATAC bus terminal. Date interchange with the PC can be implemented as one of the systems mentioned above.

C. Alternatives to the S100 System

An immediate implementation of this solution is to use the Trump Card coprocessor already in use as a software tool to replace the S100 system. Several advantages would be obtained from this
configuration. The Trump Card uses a 10 MHz Z8001 which currently addresses 512k bytes of RAM. The Trump Card fits inside the PC chassis, obtaining its power and peripheral support from the PC. The Trump Card receives software support from the PC by way of installable device drivers for the PC-DOS operating system. In addition, these device drivers, along with the Trump Card ZSYS operating system, provide high-speed data transfers between the Z8001-based system and the PC host. The Trump Card Schematic is presented in Appendix P.

The Trump Card would require an interface to the DATAC terminal subsystem interface just as the current S100 system does. This can be implemented as a simple piggyback cable to the Z8001 CPU socket, rather than as separate circuit. The incoming control signals from the bus terminal SSI would disable the Z8001 and allow the bus terminal to use the CPU's existing decoding and buffer systems to perform the word transfers.

D. Bus Terminal Interface Support

The extensible nature of the Trump Card allows for further enhancement of this system. Because the Trump Card is capable of performing many of the tasks previously assigned to the console computer, it is reasonable to transfer Bus Monitor tasks to the Z8001 to obtain maximum data processing speed.

An interesting question arises from the potential use of the Trump Card. Because of the higher clock speed on the Trump Card, the memory access time required by bus terminal accesses becomes a more significant portion of this CPU's processing time. At 10 MHz, 20 μs
is 200 clock cycles, and the Z8001 can do a fair amount of work in 200 clock cycles.

At some point, if the Z8001 is sufficiently loaded, the DMA access performed by the DATAC terminal on Trump Card memory could hinder the Z8001 system in its ability to efficiently perform those tasks. Should that happen, it would be desirable to provide an alternate method of transferring data from the bus terminal to the Bus Monitor. Rather than allow the terminal to DMA the Trump Card's memory, a new interface could be used, called Common Access Memory (CAM). This device would be very similar to the SIR in concept: a memory system that resides in the I/O space of a processor system, in this case, the Trump Card. The primary purpose of the CAM is to provide access to DATAC terminal data without the encumbrance of the bus terminal accesses interfering with normal processor activity.

Figure 6.1 shows a block diagram of the CAM. A 4k x 16-bit dual-port memory array is located in the I/O space of the Trump Card. The Z8001's 65535 I/O ports make this a simple matter. The CAM may be accessed by either one of the two Z-BUS ports: the bus terminal's or the Trump Card's. The CAM would include contention control circuitry so that if one system attempts to access the CAM while the other system is involved in a memory transaction, the contending system would be temporarily held off by assertion of the WAIT* signal. When the current user system has concluded its operation, the contending system would be granted access.

Rather than simply be a two-port RAM device like the current SIR, the CAM might further extend the role of interface device by
Zeus & Terminal

Terminal Memory Space

DMA Access

Access Control/Decoding

4k x 16 RAM

Access Arbitration Logic

Currently accessing system is given priority to CAM, contending system receives WAIT* to hold off access until CAM is released by current user.

Trump Card I/O Space

Access Control/Decoding

ZBUS To/From Trump Card

Figure 6.1. CAM Dual-Port Memory Configuration
including a presettable digital comparator to determine the progress of a message transmission. This mechanism is illustrated in Figure 6.2. As the transmitter id appears from the terminal, a P/ROM determines the word count for this message and loads the comparator with that value. Every time the terminal deposits a word, activity on the BUSREQ* and BUSACK* lines increments a counter. As soon as the incoming word count matches the expected value, the Z8001 is interrupted, secures the ID of the message originator, and appropriately processes the block. A watchdog timer could alert the Trump Card if the expected word count was not reached, thus signaling an error. While the CAM would enhance the performance of a Trump Card-based Bus Monitor, the concept is adaptable to almost any microprocessor-based system.

With the exception of the serial port and BlueBoard initialization and management routines, all software developed for the current Bus Monitor can be ported to the Trump Card. Language support for the Trump Card currently includes C and BASIC compilers and a structured assembler, both of which provide the communications routines with the PC. Additionally, an enhanced C compiler and Pascal are now available, should those facilities be deemed desirable.

Because of the enhanced data throughput of this proposed system, the role of the PC as a peripheral management system becomes more significant. With the capability to record many consecutive transmission frames, storage of this data for secondary reduction emerges as a valuable diagnostic tool. Such reduction could provide,
Figure 6.2. CAM End-of-Message Signaling to Trump Card II
for example, continuous system state analysis across successive frames.

These recommendations address the major shortcomings of the current Bus Monitor. The Trump Card is a stable product with a large installed base. Sweet Micro Systems, the Trump Card manufacturer, has provided extensive end-user support for hardware and software. The bottleneck of serial data transmission between the Z8000 and console systems is obviated by high speed byte transfers between PC-DOS and ZSYS, the Trump Card operating system. Consolidating the system into the PC chassis simplifies the hardware configuration by eliminating a separate chassis and power supply for the Z8000 system (figure 6.3). The high processor clock speed and large memory space available under the Z8001 make high-level language routines possible for preprocessing data prior to shipment to the PC for display and post-processing. Using the optional CAM as transient storage for messages from the bus terminal further frees the Z8000 from interruptions in its data processing as well as providing separate memory for data storage, should these features be deemed necessary. A comparison of the existing Bus Monitor with the proposed Bus Monitor II is presented in table 6.1.

Implementation of any of these recommendations can provide a Bus Monitor with greatly enhanced capabilities, particularly that of handling greater volumes of real-time data. Once the Bus Monitor is capable of dealing with all bus traffic in a manner consistent with the desired mode of operation, a comprehensive description of DATAAC bus activity is possible.
Figure 6.3. Proposed Bus Monitor II Installation in PC Chassis
<table>
<thead>
<tr>
<th>Bus Monitor I</th>
<th>Bus Monitor II</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Z8000 System Features</strong></td>
<td></td>
</tr>
<tr>
<td>S100 bus-based Z8002 microcomputer providing</td>
<td>SS62/ZBUS-based Z8001 microcomputer coprocessor for</td>
</tr>
<tr>
<td>interface to the DATAC bus</td>
<td>the IBM PC and compatible computers, providing</td>
</tr>
<tr>
<td>using a custom Z-BUS/S100 signal translator.</td>
<td>interface to the DATAC bus using the subset of the</td>
</tr>
<tr>
<td>Translator provides for bus terminal access of</td>
<td>ZBUS component interconnect used on the bus</td>
</tr>
<tr>
<td>Z8002 memory and optional capture of 16-bit</td>
<td>terminal subsystem interface.</td>
</tr>
<tr>
<td>interrupt tag. Data received from the bus is</td>
<td>Data received from the bus is placed in a two-port</td>
</tr>
<tr>
<td>sent bit-serially to an IBM PC-type microcomputer for storage, display, and reduction.</td>
<td>128-byte buffer for access by the host PC.</td>
</tr>
<tr>
<td><strong>Configuration</strong></td>
<td></td>
</tr>
<tr>
<td>6 MHz Z8002 microprocessor, 64k bytes RAM, 8k</td>
<td>10 MHz Z8001 microprocessor, 512k-2M bytes RAM</td>
</tr>
<tr>
<td>bytes RAM, 8k bytes ROM (contains system</td>
<td>(no wait states), 4k bytes ROM (used for system</td>
</tr>
<tr>
<td>initialization and monitor routines), 6 RS-232</td>
<td>initialization and low-level intersystem</td>
</tr>
<tr>
<td>serial channels (one channel used for console</td>
<td>communications), system must use console</td>
</tr>
<tr>
<td>terminal use); system must use console</td>
<td>computer system for data display and storage.</td>
</tr>
<tr>
<td>computer system for data storage. Data transfer</td>
<td>Data transfer between the Z8001 and PC systems is</td>
</tr>
<tr>
<td>between the S100 and PC systems is currently</td>
<td>limited by processor memory-access speeds.</td>
</tr>
<tr>
<td>limited to 9600 kbps.</td>
<td></td>
</tr>
<tr>
<td><strong>Software Support</strong></td>
<td></td>
</tr>
<tr>
<td>Z8000 assembly language.</td>
<td>Z8000 assembly language</td>
</tr>
<tr>
<td>(Pascal-like structures available using Y assembler), C, compiled BASIC, Pascal.</td>
<td></td>
</tr>
</tbody>
</table>
References


A. DATAC TERMINAL SSI PINOUT

This appendix lists the subset of the Zilog Z-BUS used in the DATAC terminal Subsystem Interface along with the pin assignments of the 67-contact barrel connector currently in use on DATAC systems.
### Current NASA ATOPS DATAC Terminal Subsystem Interface

<table>
<thead>
<tr>
<th>Pin Number</th>
<th>Pin Name</th>
<th>Pin Function</th>
<th>Source/Sink</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A0</td>
<td>Steady Address, 16-bit bus</td>
<td>Output, 3-state</td>
</tr>
<tr>
<td>2</td>
<td>A1</td>
<td>16-bit bus</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>A2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>A3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>A4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>A5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>A6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>A7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>A8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>A9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>A10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>A11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>A12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>A13</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>A14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>A15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>AE*</td>
<td>Address Enable</td>
<td>Output</td>
</tr>
<tr>
<td>18</td>
<td>RIV*</td>
<td>Receive Interrupt Vector Strobe</td>
<td>Output</td>
</tr>
<tr>
<td>19</td>
<td>WAIT</td>
<td>Wait</td>
<td>Input</td>
</tr>
<tr>
<td>20</td>
<td>XIV*</td>
<td>Transmit Interrupt Vector Strobe</td>
<td>Output</td>
</tr>
<tr>
<td>21</td>
<td>R/W*</td>
<td>Read-Write Line (Read=H, Write=L)</td>
<td>Output</td>
</tr>
<tr>
<td>22</td>
<td>AS*</td>
<td>Address Strobe used with ADO-AD15</td>
<td>Output</td>
</tr>
<tr>
<td>23</td>
<td>BUSREQ</td>
<td>Bus Request</td>
<td>Output (open collector)</td>
</tr>
<tr>
<td>24</td>
<td>DS*</td>
<td>Data Strobe used with ADO-AD15</td>
<td>Output</td>
</tr>
<tr>
<td>25</td>
<td>BUSACK*</td>
<td>Bus Acknowledge</td>
<td>Input</td>
</tr>
<tr>
<td>26</td>
<td>16 MHz</td>
<td>16 MHz Clock</td>
<td>Output</td>
</tr>
<tr>
<td>27</td>
<td>1WAIT</td>
<td>Add 1 wait state to every SSI cycle</td>
<td>Input</td>
</tr>
<tr>
<td>28</td>
<td>4 MHz</td>
<td>4 MHz clock</td>
<td>Output</td>
</tr>
<tr>
<td>Pin Number</td>
<td>Pin Name</td>
<td>Pin Function</td>
<td>Source/Sink</td>
</tr>
<tr>
<td>------------</td>
<td>---------</td>
<td>--------------------</td>
<td>--------------------</td>
</tr>
<tr>
<td>30</td>
<td>A0</td>
<td>Address/Data Bus</td>
<td>Bidirectional,</td>
</tr>
<tr>
<td>32</td>
<td>A1</td>
<td>Z-BUS compatible</td>
<td>3-state</td>
</tr>
<tr>
<td>34</td>
<td>A2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>36</td>
<td>A3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>38</td>
<td>A4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>A5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>42</td>
<td>A6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>44</td>
<td>A7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>46</td>
<td>A8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>48</td>
<td>A9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>A10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>52</td>
<td>A11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>54</td>
<td>A12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>56</td>
<td>A13</td>
<td></td>
<td></td>
</tr>
<tr>
<td>58</td>
<td>A14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>60</td>
<td>A15</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**NOTE:** The 16-bit Interrupt Vectors (XIV, RIV) are available only on a 57-pin auxiliary connector on the DATAC terminal and are not formally part of the Subsystem Interface definition.
B. ROCKWELL R6522 VIA PRODUCT DESCRIPTION

The internal register configuration of the 6522 Versatile Interface Device used to provide the simulated bus terminal SSI is presented, along with a functional description and programming guidelines.
The R6522 Versatile Interface Adapter (VIA) is a very flexible I/O control device. In addition, this device contains a pair of very powerful 16-bit interval timers, a serial-to-parallel/parallel-to-serial shift register and input data latching on the peripheral ports. Expanded handshaking capability allows control of bidirectional data transfers between VIA’s in multiple processor systems.

Control of peripheral devices is handled primarily through two 8-bit bidirectional ports. Each line can be programmed as either an input or an output. Several peripheral I/O lines can be controlled directly from the interval timers for generating programmable frequency square waves or for counting externally generated pulses. To facilitate control of the many powerful features of this chip, an interrupt flag register, an interrupt enable register and a pair of function control registers are provided.

ORDERING INFORMATION

<table>
<thead>
<tr>
<th>Part Number:</th>
<th>R6522</th>
</tr>
</thead>
</table>

- **Temperature Range**: Blank = 40°C to +70°C
- **E** = 40°C to +85°C
- **Package**: C = Ceramic
- **P** = Plastic
- **Frequency**: No Label = 1 MHz
- **A** = 2 MHz

- **Two 8-bit bidirectional I/O ports**
- **Two 16-bit programmable timer counters**
- **Serial data port**
- **TTL compatible**
- **CMOS compatible peripheral control lines**
- **Expanded “handshake” capability allows positive control of data transfers between processor and peripheral devices**
- **Latched output and input registers**
- **1 MHz and 2 MHz operation**
- **Single +5V power supply**

8522 Pin Configuration
R6522 Versatile Interface Adapter (VIA)

INTERFACE SIGNALS

RESET (RES)
A low reset (RES) input clears all R6522 internal registers to logic 0 (except T1, the registers are cleared and the Shift Register). This places all peripheral interface lines in the input state, disables the timers, shift register, etc. and disables interrupting from the chip.

INPUT CLOCK (PHASE 2)
The input clock is the system $2$ clock and triggers all data transfers between processor bus and the R6522.

READ/WRITE (R/W)
The direction of the data transfers between the R6522 and the system processor is controlled by the R/W line in conjunction with the CS1 and CS2 inputs. When R/W is low, (write operation) and the R6522 is selected, data is transferred from the processor bus into the selected R6522 register. When R/W is high, (read operation) and the R6522 is selected, data is transferred from the selected R6522 register to the processor bus.

DATA BUS (D0–D7)
The eight bidirectional data bus lines transfer data between the R6522 and the system processor bus. During read cycles, the contents of the selected R6522 register are placed on the data bus lines. During write cycles, these lines are high-impedance inputs and data is transferred from the processor bus into the selected register. When the R6522 is not selected, the data bus lines are high-impedance.

CHIP SELECTS (CS1, CS2)
The two chip select inputs are normally connected to processor address lines either directly or through decoding. The selected R6522 register is accessed when CS1 is high and CS2 is low.

REGISTER SELECTS (RS0–RS3)
The coding of the four Register Select inputs selects one of the 16 internal registers of the R6522, as shown in Table 1.

INTERRUPT REQUEST (IRQ)
The Interrupt Request output goes low whenever an internal interrupt flag is set and the corresponding interrupt enable bit is a logic 1. This output is open-drain to allow the interrupt request signal to be wire-OR'd with other equivalent signals in the system.

PERIPHERAL PORT A (PAO–PA7)
Port A consists of eight lines which can be individually programmed to act as inputs or outputs under control of Data Direction Register A. The polarity of output pins is controlled by an Output Register and input data may be latched into an internal register under control of the CA1 line. All of these modes of operation are controlled by the system processor through the internal control registers. These lines represent one standard TTL load in the input mode and will drive one standard TTL load in the output mode. Figure 2 illustrates the output circuit.

![Diagram of R6522 VIA Interface Signals](Image)

Figure 1. R6522 VIA Interface Signals
PORT A CONTROL LINES (CA1, CA2)

The two Port A control lines act as interrupt inputs or as handshake outputs. Each line controls an internal interrupt flag with a corresponding interrupt enable bit. In addition, CA1 controls the latching of data on Port A input lines. CA1 is a high-impedance input only while CA2 represents one standard TTL load in the input mode. CA2 will drive one standard TTL load in the output mode.

PORT B (PB0–PB7)

Peripheral Port B consists of eight bidirectional lines which are controlled by an output register and a data direction register in much the same manner as the Port A. In addition, the polarity of the PB7 output signal can be controlled by one of the timers while the second timer can be programmed to count pulses on the PB6 pin. Port B lines represent one standard TTL load in the input mode and will drive one standard TTL load in the output mode. In addition, they are capable of sourcing 1.0 mA at 1.5 VDC in the output mode to allow the outputs to directly drive Darlington transistor circuits. Figure 3 is the circuit schematic.

PORT B CONTROL LINES (CB1, CB2)

The Port B control lines act as interrupt inputs or as handshake outputs. As with CA1 and CA2, each line controls an interrupt flag with a corresponding interrupt enable bit. In addition, these lines act as a serial port under control of the Shift Register. These lines represent one standard TTL load in the input mode and will drive one standard TTL load in the output mode. CB2 can also drive a Darlington transistor circuit, however, CB1 cannot.

Table 1. R6522 Register Addressing

<table>
<thead>
<tr>
<th>Register Number</th>
<th>RS Coding</th>
<th>Register Description</th>
<th>Write (R/W = L)</th>
<th>Read (R/W = H)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0 0 0 0</td>
<td>Output Register B</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0 0 0 1</td>
<td>ORA, IFA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>0 0 1 0</td>
<td>DDRB Data Direction B</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0 0 1 1</td>
<td>DORA</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0 1 0 0</td>
<td>T1C-L T1 Low-Order Latches</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0 1 0 1</td>
<td>T1C-H T1 High-Order Counter</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>0 1 1 0</td>
<td>T1L-L T1 Low-Order Latches</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>0 1 1 1</td>
<td>T1L-H T1 High-Order Latches</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>1 0 0 0</td>
<td>T2C-L T2 Low-Order Latches</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>1 0 0 1</td>
<td>T2C-H T2 High-Order Counter</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>1 0 1 0</td>
<td>BR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>1 1 0 0</td>
<td>ACR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>1 1 0 1</td>
<td>PCR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>1 1 1 0</td>
<td>IFR</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>1 1 1 1</td>
<td>IER</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>1 1 1 1</td>
<td>ORA, IFA</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

NOTE: "Same as Register 1 except no handshake"

![Figure 2. Port A Output Circuit](image2.png)

![Figure 3. Port B Output Circuit](image3.png)
FUNCTIONAL DESCRIPTION

The internal organization of the R6522 VIA is illustrated in Figure 4.

PORT A AND PORT B OPERATION

The R6522 VIA has two 8-bit bidirectional I/O ports (Port A and Port B) and each port has two associated control lines.

Each 8-bit peripheral port has a Data Direction Register (DDRA, DDRB) for specifying whether the peripheral pins are to act as inputs or outputs. A 0 in a bit of the Data Direction Register causes the corresponding peripheral pin to act as an input. A 1 causes the pin to act as an output.

Each peripheral pin is also controlled by a bit in the Output Register (ORA, ORB) and the Input Register (IRA, IRB). When the pin is programmed as an output, the voltage on the pin is controlled by the corresponding bit of the Output Register. A 1 in the Output Register causes the output to go high, and a "0" causes the output to go low. Data may be written into Output Register bits corresponding to pins which are programmed as inputs. In this case, however, the output signal is unaffected.

Reading a peripheral port causes the contents of the Input Register (IRA, IRB) to be transferred onto the Data Bus. With input latching disabled, IRA always reflects the levels on the PA pins. With input latching enabled, IRA will reflect the levels on the PA pins at the time the latching occurred (via CA1).

The IRB register operates similar to the IRA register. However, for pins programmed as outputs there is a difference. When reading IRA, the level on the pin determines whether a 0 or a 1 is sensed. When reading IRB, however, the bit stored in the output register, ORB, is the bit sensed. Thus, for outputs which have large loading effects and which pull an output "1" down or which pull an output "0" up, reading IRA may result in reading a "0" when a "1" was actually programmed, and reading a "1" when a "0" was programmed. Reading IRB, on the other hand, will read the "1" or "0" level actually programmed, no matter what the loading on the pin.

Figures 5 through 8 illustrate the formats of the port registers. In addition, the input latching modes are selected by the Auxiliary Control Register (Figure 14).

Figure 4. R6522 VIA Block Diagram
HANDSHAKE CONTROL OF DATA TRANSFERS

The R6522 allows positive control of data transfers between the system processor and peripheral devices through the operation of "handshake" lines. Port A lines (CA1, CA2) handshake data on both a read and write operation while the Port B lines (CB1, CB2) handshake on a write operation only.

Read Handshake

Positive control of data transfers from peripheral devices into the system processor can be accomplished very effectively using Read Handshaking. In this case, the peripheral device must generate the equivalent of a "Data Ready" signal to the processor signifying that valid data is present on the peripheral port. This signal normally interrupts the processor, which then reads the data, causing generation of a "Data Taken" signal. The peripheral device responds by making new data available. This process continues until the data transfer is complete.

In the R3522, automatic "Read" Handshaking is possible on the Peripheral A port only. The CA1 interrupt input pin accepts the "Data Ready" signal and CA2 generates the "Data Taken" signal. The "Data Ready" signal will set an internal flag which may interrupt the processor or which may be polled under program control. The "Data Taken" signal can either be a pulse or a level which is set low by the system processor and is cleared by the "Data Ready" signal. These options are shown in Figure 9 which illustrates the normal Read Handshake sequence.

**Figure 5. Output Register B (ORB), Input Register B (IRB)**

**Figure 6. Output Register A (ORA), Input Register A (IRA)**

**Figure 7. Data Direction Register B (DDRB)**

**Figure 8. Data Direction Register A (DDRA)**
Write Handshake

The sequence of operations which allows handshaking data from the system processor to a peripheral device is very similar to that described for Read Handshaking. However, for Write Handshaking, the R6522 generates the "Data Ready" signal and the peripheral device must respond with the "Data Taken" signal. This can be accomplished on both the PA port and the PB port on the R6522. CA2 or CB2 act as a "Data Ready" output in either the handshake mode or pulse mode and CA1 or CB1 accept the "Data Taken" signal from the peripheral device, setting the interrupt flag and clearing the "Data Ready" output. This sequence is shown in Figure 10.

Selection of operating modes for CA1, CA2, CB1, and CB2 is accomplished by the Peripheral Control Register (Figure 11).

![Figure 9. Read Handshake Timing (Port A, Only)](image)

![Figure 10. Write Handshake Timing](image)

![Figure 11. Peripheral Control Register (PCR)](image)
There are two independent 16-bit counter/timers (called Timer 1 and Timer 2) in the R6522. Each timer is controlled by writing bits into the Auxiliary Control Register (ACR) to select the mode of operation (Figure 14).

**Timer 1 Operation**

Interval Timer T1 consists of two 8-bit latches (Figure 12) and a 16-bit counter (Figure 13). The latches store data which is to be loaded into the counter. After loading, the counter decrements at 8 clock rates. Upon reaching zero, an interrupt flag is set, and IRQ goes low if the T1 interrupt is enabled. Timer 1 then disables any further interrupts, or automatically transfers the contents of the latches into the counter and continues to decrement. In addition, the timer may be programmed to invert the output signal on a peripheral pin (PB7) each time it "times-out". Each of these modes is discussed separately below.

Note that the processor does not write directly into the low-order counter (T1C-L). Instead, this half of the counter is loaded automatically from the low order latch (T1L-L) when the processor writes into the high order counter (T1C-H). In fact, it may not be necessary to write to the low order counter in some applications since the timing operation is triggered by writing to the high order latch.

**Figure 12. Timer 1 (T1) Latch Registers**

**Figure 13. Timer 1 (T1) Counter Registers**
Timer 1 One-Shot Mode

The Timer 1 one-shot mode generates a single interrupt for each timer load operation. As with any interval timer, the delay between the "write T1C-H" operation and generation of the processor interrupt is a direct function of the data loaded into the timing counter. In addition to generating a single interrupt, Timer 1 can be programmed to produce a single negative pulse on the PB7 peripheral pin. With the output enabled (ACR7=1) a "write T1C-H" operation will cause PB7 to go low. PB7 will return high when Timer 1 times out. The result is a single programmable width pulse.

T1 interrupt flag will be set, the IRQ pin will go low (interrupt enabled), and the signal on PB7 will go high. At this time the counter will continue to decrement at system clock rate. This allows the system processor to read the contents of the counter to determine the time since interrupt. However, the T1 interrupt flag cannot be set again unless it has been cleared as described in this specification.

Timing for the R6522 interval timer one-shot modes is shown in Figure 15. In the one-shot mode, writing into the T1L-H has no effect on the operation of Timer 1. However, it will be necessary to assure that the low order latch contains the proper data before initiating the count-down with a "write T1C-H" operation. When the processor writes into the high order counter (T1C-H), the T1 interrupt flag will be cleared, the contents of the low order latch will be transferred into the low order counter, and the timer will begin to decrement at system clock rate. If the PB7 output is enabled, this signal will go low on the a2 following the write operation. When the counter reaches zero, the T1 interrupt flag will be set, the IRQ pin will go low (interrupt enabled), and the signal on PB7 will go high. At this time the counter will continue to decrement at system clock rate. This allows the system processor to read the contents of the counter to determine the time since interrupt. However, the T1 interrupt flag cannot be set again unless it has been cleared as described in this specification.
Timer 1 Free-Run Mode

The most important advantage associated with the latches in T1 is the ability to produce a continuous series of evenly spaced interrupts and the ability to produce a square wave on PB7 whose frequency is not affected by variations in the processor, processor response time. This is accomplished in the "free-running" mode.

In the free-running mode, the interrupt flag is set and the signal on PB7 is inverted each time the counter reaches zero. However, instead of continuing to decrement from zero after a time-out, the timer automatically transfers the contents of the latch into the counter (16 bits) and continues to decrement from there. The interrupt flag can be cleared by writing T1CH, by reading T1CL, or by writing directly into the latch as described later. However, it is not necessary to rewrite the timer to enable setting the interrupt flag on the next time-out.

All interval timers in the R6522 are "re-triggerable." Rewriting the counter will always re-initialize the time-out period. In fact, the time-out can be prevented completely if the processor continues to rewrite the timer before it reaches zero. Timer 1 will operate in this manner if the processor writes into the high order counter (T1CH). However, by loading the latches only, the processor can access the timer during each down-counting operation without affecting the time-out in process. Instead, the data loaded into the latches will determine the length of the next time-out period.

This capability is particularly valuable in the free-running mode with the output enabled. In this mode, the signal on PB7 is inverted and the interrupt flag is set with each time-out. By responding to the interrupts with new data for the latches, the processor can determine the period of the next half cycle during each half cycle of the output signal on PB7. In this manner, very complex waveforms can be generated.

A precaution to take in the use of PB7 as the timer output concerns the Data Direction Register contents for PB7. Both DDRB bit 7 and ACR bit 7 must be 1 for PB7 to function as the timer output. If one is 1 and the other is 0, then PB7 functions as a normal output pin, controlled by ORB bit 7.

Timer 2 Operation

Timer 2 operates as an interval timer (in the "one-shot" mode only), or as a counter for counting negative pulses on the PB6 peripheral pin. A single control bit in the Auxiliary Control Register selects between these two modes. This timer is comprised of a "write only" lower-order latch (T2L-L), a "read only" lower-order counter (T2L-H) and a read/write high order counter (T2H-H). The counter registers act as a 16-bit counter which decrements at 12 rate. Figure 17 illustrates the T2 Latch/Counter Registers.

Timer 2 One-Shot Mode

As an interval timer, T2 operates in the "one-shot" mode similar to Timer 1. In this mode, T2 provides a single interrupt for each write T2CH operation. After timing out, the counter will continue to decrement. However, setting of the interrupt flag is disabled after initial time-out so that it will not be set by the counter decrementing again through zero. The processor must rewrite T2CH to enable setting of the interrupt flag. The interrupt flag is cleared by reading T2CL or by writing T2CH. Timing for this operation is shown in Figure 18.

Timer 2 Pulse Counting Mode

In the pulse counting mode, T2 counts a predetermined number of negative-going pulses on PB6. This is accomplished by first loading a number into T2. Writing into T2CH clears the interrupt flag and allows the counter to decrement each time a pulse is applied to PB6. The interrupt flag is set when T2 counts down past zero. The counter will then continue to decrement with each pulse on PB6. However, it is necessary to rewrite T2CH to allow the interrupt flag to set on a subsequent time-out. Timing for this mode is shown in Figure 19. The pulse must be low on the leading edge of 02.
**R6522**

Versatile Interface Adapter (VIA)

**Figure 17. Timer 2 (T2) Latch/Counter Registers**

- **WRITE T2CN:**
  - N
  - N-1
  - N-2
  - N-3
  - 0
  - N
  - N-1
  - N-2
  - N-3

**WRITE T2CN**

**READ:**

- **BYTES LOADED INTO LOW ORDER LATCH**
- **BYTES FROM LOW ORDER COUNTER TRANSFERRED TO MPU**
- **T2 INTERRUPT**
- **FLAG IS RESET**

- **BYTES LOADED INTO HIGH ORDER COUNTER**
- **LOW ORDER LATCH**
- **COUNTER IN ADDITION TO LOW ORDER COUNTER**
- **TRANSFERRED TO MPU**

**Figure 18. Timer 2 One-Shot Mode Timing**

**WRITE T2CN**

**Operation**

**P86 INPUT**

**READ**

**IR0 OUTPUT**

**Figure 19. Timer 2 Pulse Counting Mode**

N | N-1 | N-2 | 1 | 0 | -1
SHIFT REGISTER OPERATION

The Shift Register (SR) performs serial data transfers into and out of the CB2 pin under control of an internal modulo-8 counter. Shift pulses can be applied to the CB1 pin from an external source or, with the proper mode selection, shift pulses generated internally will appear on the CB1 pin for controlling external devices.

The control bits which select the various shift register operating modes are located in the Auxiliary Control Register. Figure 20 illustrates the configuration of the SR data bits and Figure 21 shows the SR control bits of the ACR.

SR Mode 0 — Disabled

Mode 0 disables the Shift Register. In this mode the microprocessor can write or read the SR and the SR will shift on each CB1 positive edge shifting the value on CB2. In this mode the SR interrupt flag is disabled (held to a logic 0).

SR Mode 1 — Shift In Under Control of T2

In mode 1, the shifting rate is controlled by the low order 8 bits of T2 (Figure 22). Shift pulses are generated on the CB1 pin to control shifting in external devices. The time between transitions of this output clock is a function of the system clock period and the contents of the low order T2 latch (N).

The shifting operation is triggered by the read or write of the SR if the SR flag is set in the IFR. Otherwise the first shift will occur at the next time-out of T2 after a read or write of the SR. Data is shifted first into the low order bit of SR and is then shifted into the next higher order bit of the shift register on the negative-going edge of each clock pulse. The input data should change before the positive-going edge of the CB1 clock pulse. This data is shifted into the shift register during the 82 clock cycle following the positive-going edge of the CB1 clock pulse. After 8 CB1 clock pulses, the shift register interrupt flag will set and IRQ will go low.

SR Mode 2 — Shift In Under a2 Control

In mode 2, the shifting rate is a direct function of the system clock frequency (Figure 23). CB1 becomes an output which generates shift pulses for controlling external devices. Timer 2 operates as an independent interval timer and has no effect on SR. The shifting operation is triggered by reading or writing the Shift Register. Data is shifted first into bit 0 and is then shifted into the next higher order bit of the shift register on the trailing edge of each CB1 clock pulse. After 8 clock pulses, the shift register interrupt flag will set and the output clock pulses on CB1 will stop.

Figure 20. Shift Register

Figure 21. Shift Register Modes

Figure 22. SR Mode 1 — Shift In Under T2 Control
Interrupt Operation

Controlling interrupts within the R6522 involves three principal operations. These are flagging the interrupts, enabling interrupts and signaling to the processor that an active interrupt exists within the chip. Interrupt flags are set in the Interrupt Flag Register (IFR) by conditions detected within the R6522 or on inputs to the R6522. These flags normally remain set until the interrupt has been serviced. To determine the source of an interrupt, the microprocessor must examine these flags in order, from highest to lowest priority.

Associated with each interrupt flag is an interrupt enable bit in the Interrupt Enable Register (IER). This can be set or cleared by the processor to enable interrupting the processor from the corresponding interrupt flag. If an interrupt flag is set to a logic 1 by an interrupting condition, and the corresponding interrupt enable bit is set to a 1, the Interrupt Request Output (IRO) will go low. IRO is an “open-collector” output which can be “wire-OR ed” with other devices in the system to interrupt the processor.

Interrupt Flag Register (IFR)

In the R6522, all the interrupt flags are contained in one register, i.e., the IFR (Figure 29). In addition, bit 7 of this register will be read as a logic 1 when an interrupt exists within the chip. This allows very convenient polling of several devices within a system to locate the source of an interrupt.

The Interrupt Flag Register (IFR) may be read directly by the processor. In addition, individual flag bits may be cleared by writing a “1” into the appropriate bit of the IFR. When the proper chip select and register signals are applied to the chip, the contents of this register are placed on the data bus. Bit 7 indicates the status of the IRO output. This bit corresponds to the logic function

\[
IRO = IFR6 \times IER6 + IFR5 \times IER5 + IFR4 \times IER4 + IFR3 \times IER3 + IFR2 \times IER2 + IFR1 \times IER1 + IFR0 \times IER0
\]

Note:

- \( \times = \text{logic AND} \)
- \( + = \text{logic OR} \)

The IFR bit 7 is not a flag. Therefore, this bit is not directly cleared by writing a logic 1 into it. It can only be cleared by clearing all the flags in the register or by disabling all the active interrupts as discussed in the next section.

Interrupt Enable Register (IER)

For each interrupt flag in IFR, there is a corresponding bit in the Interrupt Enable Register (IER) (Figure 30). Individual bits in the IER can be set or cleared to facilitate controlling individual interrupts without affecting others. This is accomplished by writing to the IER after bit 7 is set or cleared to 1. In this case, each 1 in bits 6 through 0 clears the corresponding bit in the Interrupt Enable Register. For each zero in bits 6 through 0, the corresponding bit is unaffected.

Selected bits in the IER can be set by writing to the IER with bit 7 in the data word set to 1. In this case, each 1 in bits 6 through 0 will set the corresponding bit. For each zero, the corresponding bit will be unaffected. This individual control of the setting and clearing operations allows very convenient control of the interrupts during system operation.

In addition to setting and clearing IER bits, the contents of this register can be read at any time. Bit 7 will be read as a logic 1, however.
C. DTERM3: DATA TERMINAL SSI SIMULATION

A 68000 assembly language program provides a simulated data stream to the Bus Monitor development system to test the response of the Bus Monitor to data appearing from a bus terminal.
ORG $900
BRA PSTART

DATA terminal simulation using Motorola M68KECB
simulation sends predictable data to the 28002-based
DATA Bus Monitor Unit to test its ability to handle
data strings of increasing length.

PORT ADDRESS ASSIGNMENTS OF THE SSI SIMULATION

T_DATA EQU $30000 ; TERMINAL DATA
T_ADDR EQU $30002 ; TERMINAL ADDRESS
DORB EQU $30004 ; DATA DIRECTION REG FOR PORT B
DORA EQU $30006 ; SAME FOR PORT A

58230 TIMER INTERRUPT HANDLER ADDRESSES

INTABL EQU $100 ; ADDRESS FOR INT64 ROUTINE
TIVR EQU $10023 ; TIMER INTERRUPT VECTOR
TCR EQU $10021 ; TIMER CONTROL REGISTER
CPRH EQU $10027 ; COUNTER PRESET HIGHLBYTE
CPRM EQU $10029 ; COUNTER PRESET MIDDLE BYTE
CPRL EQU $10028 ; COUNTER PRESET LOWBYTE
TSR EQU $10035 ; TIMER STATUS REGISTER

THE FOLLOWING ARE THE ADDRESS DECODES FOR THE CONTROL SIGNAL GENERATION
OUTPUT FROM THIS DECODING IS ROUTED TO ONE-SHOTS TO PRODUCE PULSES OF THE
APPROPRIATE LENGTH

BUSREQO ON EQU $20000 ; ASSERTION OF BUSREQ*, M68KECB E1 DECODE
BUSREQOF EQU $40000 ; DEASSERTION OF BUSREQ*, E2 DECODE
INTSTRB EQU $50000 ; ASSERTION OF INTERRUPT STROBE IS*
DATASTRB EQU $60000 ; ASSERTION OF DATA STROBE DS*
RDWR EQU $70000 ; ASSERTION OF WRITE* PORTION OF R/W*

TRAP 14 FUNCTION NUMBERS FOR CONSOLE I/O MANAGEMENT

OUTPUT EQU 243 ; PORT 1 STRING OUTPUT (NO CR-LF)
OUT1CR EQU 227 ; PORT 1 STRING OUTPUT ROUTINE (W/ CR-LF)
PORTIN1 EQU 241 ; PORT 1 STRING INPUT ROUTINE
INCH EQU 247 ; PORT 1 SINGLE-CHARACTER INPUT ROUTINE

SUBROUTINES...

ASC2HEX:

FIX FOR TRAP 14. #226. TAKES AN ASCII # STRING AND CONVERTS IT INTO AN (UNSIGNED)
LONGWORD. IF AN INVALID DECIMAL DIGIT IS IN THE STRING OR IF OVERFLOW OCCURS
A LONG ZERO IS RETURNED. ENTER WITH A5 = START ADDRESS OF STRING, A6 = END ADDRESS OF STRING + 1. LONG VALUE IS RETURNED IN DO A5 IS LEFT = A6, D1 IS DESTROYED.

ASC2HEX
CLR.L D0 ;CLEAR ACCUMULATORS
MOVE.L D0,D1 ;USED IN ASCII-TO-HEX CONVERSION

NXTDGIT
MOVE.B (A5),D1 ;SET A CHARACTER FROM STRING
SUBI $30,D1 ;ADJUST TO RANGE OF 0-9
BMI NUMERR ;IF CHR WAS LESS THAN $30, BAIL OUT
CPI.B #9,D1 ;IS CHARACTER > 9?
BGT NUMERR ;IF SO, BAIL OUT WITH LONG ZERO

ADD.B D1,D0 ;ADD IN CURRENT NUMBER
ADDQ $1,A5 ;ON TO NEXT CHARACTER
CPI A5,A6 ;AT END OF NUMBER STRING?
BEQ.S GOTNUM ;IF SO, GO HOME WITH CONSTRUCTED NUMBER
MULU $10,D0 ;IF NOT, ADJUST DECIMAL DIGIT FOR ITS POWER OF 10
BVS.S NUMERR ;IF OVERFLOW, EXIT...
BRA.S NXTDGIT ;OTHERWISE, PROCESS ALL DIGITS IN STRING...

GOTNUM
RTS ;RETURN WITH HEX # IN D0

NUMERR
CLR.L D0 ;D0 = 0 IS ERROR CONDITION
RTS ;AND TAKE IT HOME

BEGINNING OF MAIN ROUTINE

PSTART
:SET UP 68230 TIMER TO PROVIDE A FRAME SYNC INTERRUPT EVERY 10/50 ms
MOVE.B $40,TIVR ;LOAD INTERRUPT VECTOR #
MOVE.L $188360,01 ;LOAD TIMER VALUE FOR 50 ms
MOVE.L #CPRH,A1 INTO TIMER PRESET REGISTER(S)
MOVEP.L D1,0(A1) LIKE SO...

MOVE.L #NXTLABEL,INTABL ;SET IRQ SVC VECTOR TO HANDLE TIMER IRQ

USER PROMPT SEQUENCE FOR MODE A DATA STRING GENERATION
:PROMPT FOR # OF DATA WORDS: "A1>"

MOVE.L #MSG1,A5
MOVE.L #MSG1+3,A6 ;ADDRESS OF LAST BYTE IN MSG
MOVE.B #OUTPUT,D7 ;FUNCTION # TO SEND STRING TO PORT 1 W/CR-LF AT END
TRAP #14 ;OUTPUT MESSAGE

MOVE.L #INBUF,A5 ;INPUT STRING BUFFER FOR # OF WORDS
MOVE.L A5,A6
MOVE.B #PORTIN1,D7
TRAP #14

BSR ASC2HEX
CMPI #0,00
BEQ OUT

;OTHERWISE

MOVE.L D0,05

;PORT 1 INPUT ROUTINE

;GET NUMBER OF WORDS PER LABEL PER FRAME

;ASCII TO HEX CONVERSION ROUTINE

;WAS A NON-NUMERIC RECEIVED?

;IF SO, RETURN TO TUTOR

;OTHERWISE

;D5 CONTAINS NUMBER OF WORDS IN HEX FOR PASSAGE TO GEN ROUTINE...

;DATA GENERATION ROUTINE

;DATA FOR TRANSFER TO Z8002 IS SETUP IN MEMORY PRIOR TO TRANSMISSION TO

;BUS MON TO AVOID ANY PROCESSING DELAYS

;EACH Ma/Mi FRAME USES 4n+1 WORDS, WHERE n IS THE NUMBER OF STATIONS TxING

;4 WORDS ARE USED TO DESCRIBE THE MESSAGE WHICH IS SIMULATED FROM EACH SYSTEM

;DATA WORDS ARE IN THE FORM OF SFFF:

WHERE

S - THE BUS STATION/SYSTEM NUMBER, 1-5

FFF - THE FRAME NUMBER, 0-FFF HEX (0-4095 DECIMAL)

;A TYPICAL SEQUENCE MIGHT LOOK LIKE THIS:

0001 SYSTEM ID NUMBER IS "0001"
0100 THE ADDRESS USED IN THE ZBUS TRANSFER IS HEX 0100
0020 20 HEX OR 32 DECIMAL WORDS ARE TO BE TRANSFERRED
1044 SAMPLE DATUM: FRAME 44 (HEX), SYSTEM 1 IS THE ORIGIN

;IF THIS WERE THE LAST SYSTEM MESSAGE SIMULATED FOR THIS FRAME, THEN THIS MESSAGE

;BLOCK WOULD BE FOLLOWED BY THIS "END OF FRAME" WORD TO SIGNAL A WAIT FOR THE

;NEXT FRAME TO BEGIN:

FFFF END OF FRAME WORD, VERY EASY TO TEST FOR THIS VALUE...

;BEGIN WITH MODE MULTIPLIER IN DO

MOVE D4,D1 ;(D1)=4 FOR An, 24 FOR SCAT-C

MOVE D1,MODESAVE ;SAVE MODE OF THIS SIMULATION
MOVE.L #WTABLE,A0
MOVE.L #TTABLE,A1

SETCT

MOVE (A0)+,D2 ;GET A WEIGHT
MULU D0,D2 ;CALC COUNT
SUBQ #1,D2 ;RECALL COUNT RANGE IS 0-n FOR n+1 ITEMS
MOVE D2,(A1)+ ;AND SAVE IT
DBF D1,SETCT ;PROCESS THE ENTIRE TABLE
CLR.L D0 ;ACCUM FOR FRAME #

MOVE.L #$1000, D0 ;DATUM TEMPLATE FOR STATION 1 (ID 0)
ADD.W D0, D1 ;ADD FRAME # TO DATUM

NEWFRM

MOVE.W 1$1000.01 ;DATUM TEMPLATE FOR STATION 1 (ID 0)
MOVE.L ICTTABLE, A1 ;POINT TO GET # OF WORDS

SAMEFRM

MOVE W1, D2 ;NEED W1 FOR INDEXED MODE
MOVE ADDTABLE(A2), (AO)+ ;SAVE BUSMON LOAD ADDRESS

MOVE.L 00, D0 ;06 IS ACCUM; DONT MUK WITH FRAME #
ADD.W 1, D0 ;5ET MODE: MAKE (07)=1 FOR MODE A, 5 FOR SCAT-C

DIVU D7, D6 ;D6 = D6/D7; D6 = REMAINDER|QUOTIENT

SWAP D6 ;D6 LO GETS OF MiF TO USE IN LOOKUP
MULU D6, D6 ;5 WORDS PER MiF TABLE ENTRY...

ADD D6, A2 ;FORM ABS OFFSET INTO CTTABLE

MOVE CTTABLE(A2), (AO)+ ;SAVE WORD CT FOR Tx OP
MOVE D1, (AO)+ ;SAVE DATUM

ADD.W #$1000, D0 ;ON TO NEXT STATION
CMP.W #$5FF, D0 ;PAST LAST ALLOWABLE DATUM?
BSE SAMEFRM ;IF NOT, CONT WITH THIS FRAME

MOVE #$FFF, (A0)+ ;EOF=RUN FLAG....
ADDQ #$1, D0 ;ON TO NEXT FRAME #
CMP.W #$2C9, D0 ;END OF FRAMING RUN? (713 IS ALL THAT 32K ALLOWS
BSE NEWFRM ;IF NOT, DO THIS FRAME

MOVE.L A0, D1 ;SAVE LAST ADDRESS IN D1 FOR EORUN TEST

700-SOME FRAMES GNNED (MiFs IF MODE C IN EFFECT)
(D1)=LAST ADDRESS+2 IN TABLE
ALL OTHER REGISTERS FREED
AT THIS POINT, THE DATA HAS BEEN SET UP IN 68000 MEMORY FOR TIMELY TRANSMISSION TO THE 18002 SYSTEM.

MSBKECB ADDRESS DECODES SENT TO ONE-SHOTS PRODUCE THE VARIOUS CONTROL SIGNALS EXPECTED BY THE BlueBoard. SINCE THE BlueBoard MUST ACCEDE TO THE DMA REQUEST, THIS TERMINAL SIMULATION DOES NOT WAIT FOR THE BUS GRANT SIGNAL; IT PAUSES AND ASSUMES THAT THE GRANT HAS BEEN GIVEN. THE ONLY 6522 SYSTEMS USED ARE THE A AND B PORTS.

assignments for data transfer are:

D0 - save area for pointer to ports acting as the ADxx and VAxx lines
D2 - Fast constant for address table calculation
D1 - address of last entry in the data stream table
D3 - word count associated with datum
D4 - interrupt vector associated with label under which the datum appears

address registers for the transfer:

A0 - address of FF to assert the BUSREQ* signal
A1 - address of FF to deassert the BUSREQ* signal
A2 - address of one-shot to provide the interrupt strobe (IS*) signal
A3 - address of one-shot to provide the data strobe (DS*) signal
A4 - address of one shot to provide the WR* half of the R/W* line
A5 - address of the AV/AD lines of the SSI simulation
A6 - pointer into the data word table

CASCADED 6522S ARE USED TO FORM A 16-BIT PERIPHERAL
BASE ADDRESS OF 6522-PAIR BEGINS AT $30000
DEVICE 1 OCCUPIES THE LOWER HALF OF THE DATA BUS (LDS*)
DEVICE 2 OCCUPIES THE UPPER HALF (UDS*)
VIA PIN NOMENCLATURE EXAMPLE: P2B1 IS PB1 ON VIA #2

T_ADDR -> TERMINAL ADDRESS = PORTA - 16 BIT DATAC ADDRESS INFORMATION
T_DATA -> TERMINAL DATA = PORTB - 16 BIT DATAC DATUM INFORMATION

CONVENTION IS D4 = INTERRUPT VECTOR/ID TAG
D2 = DATUM
D3 = DATUM COUNT/ADDRESS ;D6 = CONSTANT $FFFF FOR EOFRAME FLAG
D7 = CONSTANT - HIGHEST STATION ID NUMBER

ON SYSTEM RESET, ALL I/O PINS HI-Z INPUTS, LOOK LIKE LOGIC 1'S

MOVE #$FFFF,DDRB ;SET UP PORT B FOR ALL OUTPUTS
MOVE #$FFFF,DDRA ;SET UP PORT A THE SAME WAY

RUNITST
MOVE.L #BUSREQ00,A0 ;Ax'S USED FOR FASTEST EXECUTION TIMES
MOVE.L #BUSREQ01,A1 ;EXECUTES IN 8 CLK CYCLES
MOVE.L #INTSTRB,A2 ;ONE READ AND ONE WRITE CYCLE...
MOVE.L #DATASTRB,A3
MOVE.L #ROWR,A4
MOVE.L #T_ADDR,A5

MOVE.L #T_DATA,D0 ;SAVE DATALINE ADDRESS FOR EXG LATER ON...
MOVE.L #DATAST,A6 ;A6 GETS SET TO BEGINNING OF DATA TABLE
AND.W #1793,SR ;ENABLE ALL INTERRUPTS
MOVE.B #$A1,TCR ;TIMER SET FREERUN,ENABLED

EOFRAME
BRA EOFRAME ;WAIT FOR FRAME SYNC TO OCCUR

;IS* TAKES US TO THIS POINT

NXTLABEL
OR.W #$0300,SR ;DISABLE IRQs DURING MxF PROCESSING

MOVE.B #'TSR ;CLEAR THE INTERRUPT Tout*
MOVE (A6)+,D4 ;GET ID TAG FROM TABLE ENTRY
BPL THISFRM ;IF NOT $FFFF, PROCEED WITH NEXT STATION
AND.W #$F8FF,SR ;RESTORE INTERRUPTS FOR FRAME SYNC IRQ
RTE ;AND RETURN TO WAIT FOR END OF FRAME SYNC

THISFRM
MOVE (A6)+,D3 ;GET WORD LOAD ADDRESS USED BY BUSMON
MOVE (A6)+,D5 ;GET COUNT VALUE FOR STORE DO-LOOP
MOVE (A6)+,D2 ;GET DATA WORD TO BE DEPOSITED

CMP.L A6,D1 ;AT THE END OF THE DATA STREAM?
BCS.S EORUN ;IF SO, BAIL OUT TO 'REPEAT' PROMPT

MOVE D4,(A5) ;ASSERT ID TAG/IS VECTOR ON ADDRESS/VECTOR LINES
MOVE D4,(A2) ;ASSERT IS* AND LATCH ID FOR 8002

TXWORD
MOVE D3,(A5) ;PUT WORD COUNT/ADDRESS ON STEADY ADDRESS LINES
EXG D0,A5 ;SET UP A5 TO POINT TO THE DATA LINES
MOVE D2,(A5) ;PLACE DATUM ON SSI DATA LINES
MOVE D2,(A0) ;ASSERT BUSREQ TO ACQUIRE 8002 BUS FOR TRANSFER

;HERE, WE CORRECTLY ASSUME THAT THE BUS IS GRANTED TO US...

GOTIT
MOVE D4,(A4) ;ASSERT W* OF R/W*
MOVE D4,(A3) ;ASSERT DS*

MOVE D4,(A1) ;DEASSERT BUSREQ*
EXG     D0,AS :RESTORE AS TO POINT TO ADDRESS/VECTOR LINES

;AT THIS POINT, WE HAVE TRANSFERRED A WORD FROM THE 68000 TERMINAL SIMULATION
;TO THE Z8002'S MEMORY...

;ADDQ    #1,D3 ;READY THE NEXT WORD LOCATION COUNTER
DBF    D5, TXWORD ;LOOP TO SEND OUT (DS) WORDS

;BRA.S   NXTLABEL ;AND MOVE ON TO NEXT LABEL...

;SIMULATION OF THE SPECIFIED NUMBER OF FRAMES IS COMPLETE

;OUTPUT PROMPT FOR REPEAT OF THIS RUN...

;EORUN
OR.W  #0300,SR ;DISABLE INTERRUPTS DURING REPEAT PROMPT
MOVE.L A6, LADDR ;SAVE LAST TABLE LOOKUP VALUE

;BRA   PSTART ;ASK TO CONTINUE/INIT/BAIL OUT

;OTHERWISE....

;DOUT
MOVE  #228, D7 ;SET TO RETURN TO MONITOR
TRAP  #14 ;AND GO HOME

;STORAGE FOR THE DATAC MESSAGES...

;ADDTABLE DC  $0,$80,$100,$180,$200 :CONSTANTS CONTAINING THE WORD COUNT VALUES

;WITABLE
DC  1,1,1,1 :"WEIGHTS" TO FORM CTS, MIF 1
DC  1,1,1,1 :"WEIGHTS" TO FORM CTS MIF 2
DC  1,1,1,1 :"WEIGHTS" TO FORM CTS MIF 3
DC  1,1,1,1 :"WEIGHTS" TO FORM CTS MIF 4
DC  1,1,1,1 :"WEIGHTS" TO FORM CTS MIF 5

;CTTABLE DS  25 :NUMBER OF STATIONS WORDS PER (Ma) FRAME

;MODESAVE DS  1

;TEXT FOR OPERATOR PROMPTS:

;MSG1   DC  'A1'>

;LADDR DS  2 ;LAST ADDRESS FOR DEBUGGING
INBUF DS  2 ;STORAGE FOR 4 ASCII DIGITS CONTAINING WORD COUNT VALUE

;DATADST EQU * ;STORAGE FOR THE "DATAC" STREAM SENT TO THE SILVER BOX

END
D. GENTHEX/BIN2THEX: Y ASSEMBLER TO TEKHEX FORMATTER

The BIN2THEX C program translates a Y assembler hex format file to an ASCII file conforming to the TekHex format; the GENTHEX batch file manages the user interface by providing the appropriate I/O redirection and file management.
REM GENTHEX: Y ASSEMBLER TO STANDARD TekHex FORMAT CONVERTER
REM USES THIS BATCH FILE FOR USER PROMPTS AND FILE CHECKING
REM ***
ECHO OFF
IF .%1 == . GOTO ERR1
IF %1 == ? GOTO TELL
IF EXIST %1 GOTO NOCHEK
ECHO file %1 does not exist. specify a new source filename
GOTO EXIT
:ERR1
ECHO GENTHEX requires a source file in Y assembler format and a
ECHO .THX target filename
GOTO EXIT
:TELL
ECHO the correct command format is:
ECHO GENTHEX <sourcefilename> <destfilename>
ECHO where <sourcefilename> is a Y assembler hex file and
ECHO <destfilename> is the TekHex target file
ECHO if <destfilename> exists, the current file will be appended
GOTO EXIT
:NOCHEK
IF EXIST %2 ECHO **WARNING** FILE %2 ALREADY EXISTS!!
ECHO press control C to cancel, or
PAUSE
BIN2THEX < %1.HEX >> %2.THX
:EXIT
#include "C:STDIO.H"
/* BIN2THEX.C */
/*
BIN2THEX: program to translate a Y assembler hex format file to a standard TekHex file. This file is used by a compatible host system for transfer to a target system using the standard TekHex protocol. Rather than incorporate file checking and user-prompts in the main program, BIN2THEX is intended to be run from the GENTHEX batch file which performs these services. BIN2THEX uses DOS 2.x and higher by using I/O redirection for file operations. This has the advantage of allowing debugging from the console.

Refer to figure x.x for the Y assembler hex file format
*/

main()
{

int inbyte; /* the character read from the STDIN device */
unsigned load_add; /* 16-bit value specifying the initial load address for a given .HEX file */
char add_hi, add_lo; /* these bytes contain the hi and low order bytes of the load address while being read from STDIN, this is done to facilitate debugging and doesn't appreciably slow operation */
char hi_nyb, lo_nyb, TEKHEXarray[20]; /* these bytes are used to extract the nybbles of the data from the hex file to form the two checksums used by the TekHex record; the array is used to hold the string of characters which form the record */
char add_cksum, code_cksum, I, J; /* respectively, the first and second checksums in the in the record being formed, and two miscellaneous index/counter values */

/* code begins... */

for (I = 0; I <= 13; I++) inbyte = getc(stdin); /* skip first 14 bytes of file */
add_hi = getc(stdin); /* input first byte of load address */
add_lo = getc(stdin); /* same for second byte of load address */
load_add = (add_hi << 8) + add_lo; /* load address formation */

for (I = 0; I <= 15; I++) inbyte = getc(stdin); /* skip next 16 bytes (segmentation info) of file */
while (inbyte != EOF)
code_cksum = 0; /* reset the code checksum for each record of the TEKHEX */
* file being prepared */
hi_nyb = hi_nibl(add_hi); /* split hi-order address byte to form the */
lo_nyb = lo_nibl(add_hi); /* nybble checksum of the address and byte count */
add_cksum = hi_nyb + lo_nyb; /* checksum first byte of address */
hi_nyb = hi_nibl(add_lo); /* split the low-order address byte */
lo_nyb = lo_nibl(add_lo); /* as done for the hi-order byte */
add_cksum = add_cksum + hi_nyb + lo_nyb; /* checksum second byte of address */

for (I = 0; I <= 19; I++) /* collect no more than 20 (data) bytes per record */
/* the 20-byte limit is an artificial constraint for ease of debugging. this value may */
/* be adjusted as long as the total number of characters in the record is accepted */
/* by the Z8002 TekHex receiver program */
{
    if (inbyte = getc(stdin) == EOF) break; /* get a byte */
    TEKHEXarray[I] = inbyte; /* save it for output */

    hi_nyb = hi_nibl(inbyte); /* form checksum */
    lo_nyb = lo_nibl(inbyte);
    code_cksum = code_cksum + hi_nyb + lo_nyb;
}

I = I-1; /* always adjust I from ++ of loop so is <= 19 */

if (inbyte == EOF) /* end of file and we have a little cleaning-up to do... */
{
    hi_nyb = hi_nibl(TEKHEXarray[I]); /* back up to last legitimate data byte value */
    lo_nyb = lo_nibl(TEKHEXarray[I]);

    code_cksum = code_cksum - hi_nyb - lo_nyb; /* adjust code checksum for addition of tag byte value */
    I--; /* adjust count for tag byte */
}

/* at this point, I is the (N-1) of binary images read */
hi_nyb = hi_nibl(I+1); /* add nybbles of byte count to the checksum */
lo_nyb = lo_nibl(I+1);
add_cksum = add_cksum + hi_nyb + lo_nyb;

/* now, let us output the current TEKHEX line from the info we've collected */
/* rather than doing a hex-to-ASCII conversion, we exploit the formatted output */
/* capabilities of C to have it do the conversion for us */
printf("%04x",load_add); /* print the load address */
printf("%02x",I+1); /* ... the byte count for a given record */
printf("%02x",add_cksum); /* and the first checksum */
for (J = 0; J <= I; J++)
    printf("%02x\n",TEKHEXarray[J]); /* output the bytes */

printf("%02x\n",code_cksum); /* the code byte checksum */
printf("\n "); /* and a CR-LF for the end */
/* putchar(0x00); /* this CR for debugging purposes */
    load_add = load_add + I; /* update the load address */
    load_add = load_add & 0xFF00; /* hi address byte for cksum computation */
    load_add = (load_add & 0x00FF); /* lo address byte... */
}
/* end of WHILE... */

printf("/00000000\n"); /* the tag record, indicate end of TEKHEX file */
exit(); /* end of main routine */

*/ end of main *********************************************/

hi_nibl(inbyte)
/* this function extracts the hi-order nybble from a byte and returns that value
right justified in an 8-bit field */
char inbyte;
{
    char outbyte1;
    outbyte1 = inbyte;
    outbyte1 = outbyte1 & 0xF0; /* turn XN into X0 by masking off the lower bits */
    outbyte1 = outbyte1 >> 4; /* turn X0 into 0X by shifting right 4 bits */
    return(outbyte1);
}

lo_nibl(inbyte)
/* this function extracts the low order nybble from a byte and returns it
right justified in an 8-bit field */
char inbyte;
{
    char outbyte2;
    outbyte2 = inbyte;
    outbyte2 = outbyte2 & 0x0F; /* mask off high order bits, turn NY into 0Y */
    return(outbyte2);
}
E. TEKHEX FILE FORMAT AND TRANSFER PROTOCOL

The standard TekHex file format is described, along with the TekHex transfer protocol.
TekHex Record Formats

Each TekHex record is an ASCII representation of hexadecimal data. Two ASCII characters are required to represent one digit of hex data. The loading computer knows to reconstitute the hex values from their ASCII representations. There are two types of records in the standard TekHex format: a data record and a terminator record.

Typical Data Record Format:

/AAAAANNC1DDDD...DDDC2 <CR>

where:

'AAAA' - 16-bit hexadecimal load address for this record
'NN' - 8-bit hexadecimal byte count for this record
'C1' - 8-bit nybble checksum of the load address and byte count; both checksums are the modulo-256 sum of each 4-bit hex digit represented in the string
'DDD...DDD' - Two-digit pairs representing the data being transferred; the number of bytes is equal to 'NN'
'C2' - 8-bit checksum of 'NN' bytes of load data

Terminator Record:

/AAAA00C1

where:

'AAAA' and 'C1' are as above, but the zero byte count signifies the end of the load operation.
TekHex Downloader Protocol

HOST                   TARGET
Idle, waiting for 'ACK' prompt

> Downloader starts by sending 'ACK' token to host

Until end of file: read next record from disk file and send it byte-serially to the target system, then wait for reply. If 'NAK' received: increment error counter; if error limit exceeded, abort load and display error message.' If 'ACK' received: proceed to send next record from file as above.

> Save latest record in buffer, extract load address and byte count and form first checksum. If byte count is zero: exit loader and return to monitor program. If first checksum error: send 'NAK' token to host and wait for resend of the same record. If first checksum is ok: sequentially store bytes from record beginning at specified load address, form checksum and compare with second checksum. If second checksum error: proceed as above for bad checksum. If second checksum is ok: send 'ACK' token to host and wait for a new record.

'ACK' = ASCII '1' (31 hex)
'NAK' = ASCII '7' (37 hex)
F. Z8KLOADER: BUS MONITOR TEKHEX DOWNLOADER

A segmented Z8000 assembly language program is described which interfaces the S100 system to a TekHex compatible host.
; DATAC BUS MONITOR:
; LOADER FOR Z8000 PROCESSOR INTERFACE TO DATAC SYSTEM
; AUTHOR: S.M. NOVACKI 2 SEPT 83
; REV 22 NOV 83: INCLUDES ERROR HANDLER FOR EXITS TO MONITOR

; MACRO DEFINITIONS HERE:
MACRO NYBSUM
  LDB  RL2,RH2  ;TRANSPOSE HEX DIGITS
  SRAB RL2,004  ;MAKE MOD THE LOD
  AND  R2,$00FH  ;MASK OFF HO BITS
  ADDB  RH2,RL2  ;ADD NYBBLES W/O CARRY
;RH2 HOLDS NYBBLE CHECKSUM, TRANSFERRED TO RH7
ENDM

;THIS MACRO PERFORMS A TEST FOR CHECKSUM ERRORS, IF >5 THE LOAD IS ABORTED
MACRO ERRMSG
  INC  R13  ;COUNT NEW ERROR OCCURRENCE
  CP  R13,5  ;REACH MAX# OF ERRORS?
  JR  UGT,ABRTLD  ;TOO MANY ERRORS- RETURN TO MONITOR
  SET  R12,'01  ;SET 'OLD STRING, REPEAT' FLAG
  LDB  ACKBUF,'NNAK  ;READY BAD TX MSG
  JR  NEWSTR  ;REQUEST REPEAT OF MSG AND CLEAR INBUF
ENDM

; ORG 0FE00H
; I/O STRING BUFFER DEFINITIONS, MUST BE ORG'D IN RAM
ACKBUF BLOCK 4  ;THREE BYTE BUFFER TO HANDSHAKE WITH 8550 DURING FILE TX
INBUF BLOCK 80  ;80 BYTE BUFFER FOR RECEIVING TEKHEX FILES
TKHXIN BLOCK 8
TKINAK BLOCK 8  ;I/O FC BLOX (WORKSPACE)

; ORG 080EH
;CONSTANT DEFINITIONS:
PROMPT EQU 3EH  ;8550 HANDSHAKE PROMPT CHAR
ACK EQU 30H  ;MSG RECEIVED TOKEN
NAK EQU 37H  ;MSG NOT RECEIVED TOKEN
RECMRK EQU 00H  ;CR USED TO TERMINATE PROMPT STRING
RECMRK EQU 2FH  ;'SLASH' CHAR USED TO DELIMIT TEKHEX RECORDS

; BEGINNING OF LOADER ROUTINE;
; CONSULT ZMON.DASSY AND .DUMP TO DETERMINE ACTUAL ADDRESSES
; BEGINNING OF ROMABLE ROUTINES, ALL JUMPS RELATIVE, ONLY
; RAM REFERENCES ARE ABSOLUTE FOR DURATION OF LOADER OPERATION

TMMSG ASCII 'STD TEKHEX LOADER '  ;NOTE# OF BYTES IN STRING MUST BE EVEN

INTCOM CLR ACKBUF  ;ONLY 3 OF 4 BYTES USED
CLR ACKBUF+2  ;IN HANDSHAKE SEQUENCE
LDB ACKBUF+4,'RECEND  ;READY STRING FOR
LDB ACKBUF+3,'PROMPT  ;TEK HANDSHAKE
CLR R12  ;FLAG: 0=NEW STRING, 1=REP'T OF LAST STRING
CALR SETIO ;SET UP FCB FOR INPUT OPERATIONS
    ;SET UP FCB FOR OUTPUT OPERATIONS
NEWSTR CLRb RL2 ;(R2)=0 FOR ZAPPING
LD R10,#80 ;NUMBER OF BYTES TO BE ZAPPED
LD R9,INBUF
ZAPWRD LDB R9(R10),RL2 ;ZERO OUT INBUF (I HOPE...)
DEC R10
JR NZ,ZAPWRD;
OUTMSG LD R1,#KINAK ;SELECT SIGNAL MODE FOR TEK
SC #0 ;OUTPUT PROMPT VIA MONITOR ROUTINE

;WITH A SERIAL LINE DEDICATED TO THE Z8K-TEK INTERFACE, ODDS ARE
;THERE WON'T BE ANY JUNK BEFORE THE PROMPT AND THE FIRST HEX RECORD.
;UNTIL THAT SERIAL LINE IS ESTABLISHED, WE'LL SHARE THE ONE WITH
;Z8K CONSOLE DEVICE AND PROVIDE FOR GETTING RID OF ANY BAD DATA
;WE MAY HAPPEN TO READ. ONCE A SEPARATE SERIAL LINE IS AVAILABLE, WE CAN
;DISCARD THE 'FIND START-OF-RECORD' ROUTINE

;IDLE 8550 BEGINS TO TX AFTER THE PROMPT SENT BY OUTMSG

GETSTR LD R1,#KHXIN ;SELECT HEX RECORD READ-MODE
SC #0 ;GET HEX RECORD AND SAVE IT AT INBUF
LDA R8,INBUF ;SET BASE ADDRESS OF HEX STRING

;AT THIS POINT, WE SHOULD HAVE ONE COMPLETE TEKHEX RECORD FOR PROCESSING
;REGISTER ASSIGNMENTS FOR REDUCING THE ASCII STRING
;  R1: TRANSIENT AREA FOR CONSOLE I/O
;  R2: WORK AREAS FOR CHECKSUM COMPUTATION
;  R3,R4: WORK AREAS FOR ASCII HEX CONVERSION
;  R5: WORKSPACE FOR FINDING INCOMING ASCII STRING
;  R6: CONTAINS THE LOAD ADDRESS OF THE DATA
;  RH7: CONTAINS THE NYBBLE CHECKSUNS
;  RL7: CONTAINS THE# OF DATA BYTES IN THE RECORD
;  R8: POINTER INTO ASCII STRING FOR HEX GENERATION
;  R13: CONTAINS CHECKSUM ERROR COUNT

;FIRST WE'LL SCAN FOR JUNK THAT THE Z8K MAY HAVE READ BEFORE
;THE 8550 STARTED TX OF THE HEX FILE; THIS SECTION CAN BE
;DELETED IF WE DEDICATE A SERIAL PORT FOR 8550/Z8K COMMUNICATION
;'SLASH' CHR DELIMITS START OF DATA
SEEK CP R8,INBUF+80 ;AT THE END OF THE INPUT BUFFER?
JR EQ,STREQ ;IF SO, THE WHOLE RECORD WAS JUNK,GET ANOTHER
CPB #R8,#RECMRK ;SCAN INBUF FOR THE 'SLASH' CHARACTER
JR EQ,TSTSTR ;FOUND IT!
INC R8 ;ON TO THE NEXT CHAR
JR NE,SEEK ;HEADER NOT FOUND, TRY AGAIN
STREQ LD ACKBUF,NAK ;BAD TX,ASK FOR REPEAT OF STRING
JR NEWSTR ;DO THE ASKIN'

;END OF SOH-SCANNER ROUTINE

;WE'LL ASSUME THAT A VALID RECORD HAS BEEN READ
JR N2,OLDSTR ;DON'T RESET ERROR ACCUM IF THIS IS A REPEAT
CLR R13 ;ZERO OUT CKSUM ERROR ACCUMULATOR
OLDSTR CALR CHKTRM ;SEE IF THE RECORD IS THE ZERO-LENGTH TERMINATOR
;IF TERM RECORD IS FOUND, RETURN TO MONITOR
INC R8 ;MOVE POINTER PAST HEADER TO FIRST ASCII CHARACTER
(R8)=ADDRESS OF FIRST CHAR IN HEX STRING
CALR ASCHEX ;GET 1ST BYTE OF ADDRESS
LDB RH2,RH4 ;1ST BYTE TO CKSUM ACCUMULATOR
NYBSUM
LDB RL2,RH2 ;TRANSPOSE HEX DIGITS
SRAB RL2,#04 ;MAKE HOD THE LOD
AND R2,#0F0FH ;MASK OFF HO BITS
ADDB RH2,RL2 ;ADD NYBBLES W/O CARRY
LDB RH7,RH2 ;TO CHECKSUM ACCUMULATOR
LDB RH6,RH4 ;H0BYTE OF ADDRESS TO R6
INC R8 ;NEXT DIGIT
CALR ASCHEX ;GET SECOND BYTE OF LOAD ADDRESS
LDB RH2,RH4 ;2ND BYTE TO CKSUM ACCUMULATOR
NYBSUM
LDB RL2,RH2 ;TRANSPOSE HEX DIGITS
SRAB RL2,#04 ;MAKE HOD THE LOD
AND R2,#0F0FH ;MASK OFF HO BITS
ADDB RH2,RL2 ;ADD NYBBLES W/O CARRY
ADDB RH7,RH2 ;ADD IT TO ACCUM
LDB RL6,RH4 ;LOBYTE TO R6; LOAD ADDRESS IS NOW COMPLETE
INC R8 ;ON TO THE BYTE COUNT
CALR ASCHEX ;GET# OF BYTES IN MSG
LDB RH2,RH4 ;ADD IT TO CHKSUM
NYBSUM
LDB RL2,RH2 ;TRANSPOSE HEX DIGITS
SRAB RL2,#04 ;MAKE HOD THE LOD
AND R2,#0F0FH ;MASK OFF HO BITS
ADDB RH2,RL2 ;ADD NYBBLES W/O CARRY
ADDB RH7,RH2 ;ADD RUNNING NYBBLE CHECKSUM
LDB RL7,RH4 ;SAVE# OF DATA BYTES IN HEX FOR RAM LOAD
INC R8 ;GET CHAR CNT FROM STRING
CALR CHKSUM ;TEST 1ST BYTE-CHECKSUM
JR EQ,SUMOK ;NO PROBS,GO ON
ERRMSG
INC R13 ;COUNT NEW ERROR OCCURRENCE
CP R13,#5 ;REACH MAX# OF ERRORS?
JR UGT,ABRTLD ;TOO MANY ERRORS, RETURN TO MONITOR
SET R12,#01 ;SET 'OLD STRING, REPEAT' FLAG
LDB ACKBUF,#NAK ;READY BAD TX MSG
JR NEWSTR ;REQUEST REPEAT OF MSG AND CLEAR INBUF
SUMOK CLR B RH7 ;RESET ACCUMULATOR FOR SECOND CHECKSUM
HXLLOAD INC R8 ;NXT CHR
CALR ASCHEX ;FORM DATA BYTE
LDB RH2,RH4 ;SENT TO CKSUM ACCUM
NYBSUM
PRODUCE AND COMPARE SECOND BYTE-CHECKSUM  
NO ERRORS  
TRANSPOSE HEX DIGITS  
MAKE MOD THE LOD  
MASK OFF HO BITS  
ADD NYBBLES W/O CARRY  
ADD R7,H2  
ANOTHER DIGIT TO BE SUMMED  
STORE MACHINE CODE  
NEXT RAM LOCATION...  
ONE LESS BYTE TO STORE  
UNTIL (RL7)=0, STORE THEM BYTES!

RECORD LOAD COMPLETE  
PRODUCE AND COMPARE SECOND BYTE-CHECKSUM  
NO ERRORS

166

END OF MAIN ROUTINE; HERE ARE THE SUBRoutines...

;ASCHEX: THE ASCII CHARACTERS WHOSE ADDRESSES ARE (R8) AND (R8)+1 ARE
;CONSOLIDATED TO FORM ONE HEXADECIMAL BYTE. R3 AND R4 ARE THE WORK SPACE WITH
;THE FORMED HEX BYTE LEFT IN RH4.

;GET 1ST ASCII CHARACTER
;ADJUST ASCII IF CHR IS A-F
;MASK OFF ZONE BITS
;LSBITS BECOME MSBITS
;READY FOR NEXT DIGIT
;NEXT DIGIT
;GET IT
;ADJUST ASCII IF CHR IS A-F
;PROCESS IT
;FORM COMPLETE BYTE OF DATA
;GO HOME
; CHKSUM: COMPARE THE COMPUTED CHECKSUM WITH THE VALUE CONTAINED IN THE
; STRING TRANSMITTED FROM THE 8550. RUNNING CHECKSUM IS MAINTAINED IN
; RH7. THIS ROUTINE CALLS ASCHEX TO READ THE ASCII STRING AND GEN THE
; TX CHECKSUM.

; CHKSUM: CALR ASCHEX ; SET 1ST BYTE-CHECKSUM
    CPB RH7,RH4 ; COMPARE CALCULATED AND GIVEN CHECKSUMS
    EXIT RET ; REQUEST ANOTHER TX OF THE STRING IF NEEDED

; CHKTRM: SCANS THE INPUT BUFFER FOR A BYTE COUNT OF ZERO. USES ASCHEX
; TRANSLATE THE TWO ASCII CHARACTERS OF THE DATA COUNT TO HEX. IF THE
; BYTE COUNT IS ZERO, THE LOAD IS CONCLUDED WITHOUT A CHECKSUM SCAN AND CONTROL
; IS RETURNED TO THE MONITOR
; ENTER WITH (R8) = LOCATION OF 1ST CHAR IN LOAD ADDRESS

; CHKTRM: LD R10, R8 ; SAVE CURRENT POSITION IN STRING
    INC R8, #5 ; AIM AT 1ST CHR OF BYTE COUNT
    CALR ASCHEX ; FORM BYTE COUNT
    LD R8, R10 ; RECOVER ORIGINAL POINTER
    TESTB RH4 ; IS DATA STRING LENGTH ZERO?
    RET NE ; NO, GO BACK AND FINISH PROCESSING

; AT THIS POINT, WHO CARES ABOUT A BIT-ERROR?
    LD ACKBUF, #ACK ; SIGNAL THE END
    LD R1, #TKINAK ; READY THE MSG
    SC #0 ; SIGNAL TRANSFER END TO HOST COMPUTER
    SC #3 ; LOAD COMPLETED, RETURN TO MONITOR

; SETIO: USED TO RESET FCB FOR SC0
    SETIO LD R10, #TKXIN ; DEST FOR MOVE
    LD R11, #IOBLK ; SOURCE FOR MOVE
    LD R9, #08H ; # OF WORDS TO MOVE
    WMOVE LDIR @R10, @R11, R9 ; DO IT!
    RET ; GO HOME...

; IOBLK WORD 0100H ; BLOCK RECEIVE MODE OF MONITOR CONSOLE HANDLER
    WORD 0000H ; NOT USED
    WORD INBUF ; INBUF BUFFER LOCATION
    WORD 0050H ; STRING LENGTH IS 80 DECIMAL BYTES TO ALLOW FOR JUNK

; WORD 0200H ; BLOCK TRANSMIT MODE FOR SYSTEM CALL #0
    WORD 0000H ; NOT USED
    WORD ACKBUF ; START ADDRESS OF PROMPT-ACKNOWLEDGE BUFFER
    WORD 003H ; ONE BYTE FOR PROMPT, ONE FOR ACK-NAK TOKEN, ONE FOR EOL TOKEN

; TSTNUM: CORRECTS ASCII CHARACTERS FROM A TO F TO ALLOW FOR SIMPLE
; MANIPULATION TO HEX FORM
    TSTNUM CPB RL4, #39H
    JR LE, TSTNUM ; IF 0-9, NO CORRECTION NEEDED
    ADDB RL4, #9 ; ELSE ADD OFFSET OF 9 TO PRODUCE USEABLE LO NYBBLE
    ISNUM RET ; BACK TO ASCHEX

; end of loader and subroutines

END INTOCOM; PROGRAM START ADDRESS FOR ASSEMBLER
G. SENDTHEX: TEKHEX HOST DOWNLOADER PROGRAM

This C language program provides a serial-port equipped host which can support standard C constructs with the TekHex file transfer protocol.
/*
  8550 STANDARD TekHex DOWNLOAD EMULATOR, ORIGINALLY
  WRITTEN IN AZTEC C FOR THE CP/M ENVIRONMENT
  
  - AUTHOR: S. NOVACKI
  - CREATION DATE: JULY, 1984 AS BUSLDR.C
  - CP/M VERSION REvised 1/20/86 FOR MS-DOS AS SENDTHEX.C
  - GENERIC VERSION 4/20/86
*/

#include 'stdio.h' /* standard I/O used for file handling */
/*
  this INCLUDE will vary according to the
  system configuration, compiler used, etc.
*/

#define ACK '0' /* definitions of: the ACK token */
define NAK '7' /* the NAK token */
define CR 13 /* end-of-line flag */
define CTRLZ 26 /*
infile: pointer for source file (from disk)
umchar: subscript for reading characters from disk file into buffer vector
outptr: subscript for sending buffer characters to serial port
argc: command line argument count, used by 'C' console command processor
errcount: number of consecutive reception errors
iolinebuffer: array used in moving characters from disk file (using standard
I/O) target system serial channel (using system-specific software)
reply: token read from BusMon system to indicate quality of message: ACK or NAK */

FILE *infile, *fopen();
int iolinebuffer[80], numchar, outptr, argc;
int intoken = 0,errcount = 0;
char reply = '7';

main(argc,argv)
char *argv[];
open disk file to be sent to the BUSMON system if a NULL is returned, OPEN has failed, exit to DOS

if ((infile = fopen(*++argv, "r")) == NULL)
{
  printf("open failure on file %s\n", *argv); exit(99);
}

/* get_reply(); /* get first ACK to commence file transmission */
/* when using CP/M or MS-DOS terminal emulators, it is not likely that
the operator will be able to exit his terminal simulator and then
load the TekHex downloader fast
enough to catch the BusMon sending the first ACK token, thus the
BusMon waits for the console system and the console is waiting
for the BusMon (unlike the Tek 8550 which runs the downloader
as a background communications task under DOS/50. To avoid this
potentially troublesome spot, we skirt the issue completely by not
looking for the first token, that is, we will assume that the
BusMon has sent the ACK to initiate the download process.
*/

looping:
errcount = 0;
get_line(); /* read a line from the TEKHEX disk file */
if ((char)intoken == (char)CTRLZ) goto tx_done; /* bail out as needed */

while (reply != ACK)
{
  retrans_record();
  get_reply();
  if (errcount > 5) load_error(); /* if too many errors, exit with error msg */
}

goto looping;

*/

get_line()
/* function to read < 80 character from the TEKHEX disk file */
{
for (numchar = 0; numchar < 80; ++numchar)
{
  intoken = getc(infile);
  if (intoken == CTRLZ) goto readdone;
}
iolinebuffer[numchar] = intoken;
if ((char)intoken == (char)CR) goto readdone;
}
readdone:()
}

/*****************************************************/

tx_line()
{
for (outptr = 0; outptr <= numchar; ++outptr)
{
    put_token(token_device, iolinebuffer[outptr]); /* send current line to BusMon */
    putchar(iolinebuffer[outptr]); /* echo TEKHEX file to console to monitor download */
}

/*****************************************************/

get_reply()
/* receives reply token from the BusMon unit after tx_line is performed */
/* if a NAK is received, log the error count */
{
    if ((reply = get_token(token_device)) != ACK) ++errcount;
}

/*****************************************************/

retrans_record()
/* tx_line by another name, done for improved legibility */
/* since numchar is not destroyed by tx_line, this offers a very convenient */
/* way to retransmit the same line of characters */
{
    tx_line();
}

/*****************************************************/

load_error()
/* only if five successive load errors are reported by the BusMon */
{
    printf("error limit exceeded, load operation aborted\n");
    fclose(infile); /* close the disk file */
    exit(88); /* return to OS with error code 88 */
}

/*****************************************************/

/* these dummy functions are needed to perform unbuffered I/O between the host
and target systems. Under CP/M, a simple function to access the PUN: and RDR: devices for character I/O using BDOS Calls 3 and 4 work well, but obviously this requires specific knowledge of CP/M. Alternatively, one could talk directly to the system's serial ports; similarly un C-like. If your OS resource manager allows you to do unbuffered I/O, see if your C compiler gives you the appropriate hooks. Otherwise, resort to inline code or write a ML function. */

get_token(token_device)

char *token_device;
{char c;

c = getc(token_device);
return c;
}

/***

put_token(token_device, c)

char *token_device;
char c;
{

putc(token_device,c);
}

******************************************************************************/
H. MPU 8000 II MONITOR COMMAND SUMMARY

The various monitor commands of the Ithaca Intersystems Z8002 monitor and their formats are described.
MPU8000 II Monitor Command Summary

The monitor commands are listed according to functional categories below. All commands are single letter upper case entries from the console keyboard, sometimes followed by an argument of one or more hexadecimal numbers. These numerical arguments are represented below by the expression "XXXX" for a multiple digit number or "X" for a single digit number.

Blank spaces separating command letters and any arguments are optional.

Keyboard edit functions:
  Delete last character
  Delete entire command line
  Return to monitor command mode

System edit functions:
  Display eight words of memory
    starting at address XXXX
  Display next eight words of
    memory location XXXX
  Move XXXX words of memory

Examining and modify...
  Memory location XXXX
  CPU register X
  Normal Stack Pointer
  System Stack Pointer
  Program Counter
  Flag and Control Word

Execution Functions:
  Execute program at XXXX
  Set breakpoint A at XXXX
  Set breakpoint B at XXXX
  Clear both breakpoints
  Trace single instruction
    beginning at address XXXX
  Trace the next instruction

DEL (ASCII 7FH)
^X (ASCII 18H)
ESC (ASCII 1BH)
D XXXX
D (or carriage
  return memory
  after current D
  command)
M XXXX (monitor
  will prompt for
  source and
  destination
  addresses)
E XXXX
R X
R F
Y
P
F
G XXXX
A XXXX
B XXXX
C
T XXXX
T (or carriage
  return after a T
  instruction is
  executed)
I. GENZHEX/MAKEZHEX: Y ASSEMBLER TO .ZHX FORMATTER

The MAKEZHEX program translates a Y assembler hex format file to the embedded-monitor-command .ZHX format for downloading data to the MPU 8000 II CPU. The GENZHEX batch file manages the user interface as well as provides I/O redirection.
REM GENZHEX: Y ASSEMBLER TO MPU 8000 II MONITOR FORMAT CONVERTER
REM USES THIS BATCH FILE FOR USER PROMPTS AND FILE CHECKING
REM ***
ECHO OFF
IF .%1 == . GOTO ERR1
IF %1 == ? GOTO TELL
IF EXIST %1 GOTO NOCHEK
ECHO file %1 does not exist, specify a new source filename
GOTO EXIT
:ERR1
ECHO GENZHEX requires a source filename in Y assembler format and a
ECHO .ZHX target filename
GOTO EXIT
:TELL
ECHO the correct command format is:
ECHO GENZHEX <sourcefilename> <destfilename>
ECHO where <sourcefilename> is a Y assembler hex file and
ECHO <destfilename> is the ZHX target file
ECHO if <destfilename> exists, the current file will be appended
ECHO the filenames may be prefixed by an optional pathname
GOTO EXIT
:NOCHEK
IF EXIST %2 ECHO **WARNING** FILE %2 ALREADY EXISTS!!
ECHO press control C to cancel, or
PAUSE
MAKEZHEX < %1.HEX >> %1.ZHX
:EXIT
MAKEZHEX: program to translate a Y assembler hex format file to an embedded MPU 8000 II monitor command file. This file uses the "E"xamine command of the monitor program to initiate a sequential byte-load operation on the Z8002 system. Each load byte is followed by a CR to ready the next location for the deposit operation until an ESC is encountered to terminate the procedure. Rather than incorporate file checking and user-prompts in the main program, MAKEZHEX is intended to be run from the GENZHEX batch file which performs these services. MAKEZHEX uses DOS 2.x and higher by using I/O redirection for file operations. This has the advantage of allowing debugging from the console.

Refer to figure 3.11 for the Y assembler hex file format and Appendix H. for the MPU 8000 II "Examine" command format.

```c
#include "C:STDOIO.H"
/* MAKEZHEX.C */
/*
 MAKEZHEX: program to translate a Y assembler hex format file to an embedded MPU 8000 II monitor command file. This file uses the "E"xamine command of the monitor program to initiate a sequential byte-load operation on the Z8002 system. Each load byte is followed by a CR to ready the next location for the deposit operation until an ESC is encountered to terminate the procedure. Rather than incorporate file checking and user-prompts in the main program, MAKEZHEX is intended to be run from the GENZHEX batch file which performs these services. MAKEZHEX uses DOS 2.x and higher by using I/O redirection for file operations. This has the advantage of allowing debugging from the console.

Refer to figure 3.11 for the Y assembler hex file format and Appendix H. for the MPU 8000 II "Examine" command format.
*/
main()
{
    int bytecount; /* this integer contains the number of bytes to be deposited in this .ZHX file */
    char I; /* miscellaneous index/loop counter */

    /* code begins... */
    for (I = 0; I <= 3; I++) getc(stdin); /* skip first 4 bytes of file */
    bytecount = (getc(stdin) << 8); /* read first half of byte count and left-justify */
    bytecount = (((bytecount + getc(stdin)) - 1) / 2); /* get lower half of byte count, complete count, adjust it to ignore tag byte, and make byte count a "COUNT" count for the "E" command */
    for (I = 0; I <= 7; I++) getc(stdin); /* skip next 8 bytes of file */
    putchar('E'); /* 'EXAMINE' command of monitor to start the load */
    printf("%02x",getc(stdin)); /* first byte of load address */
    printf("%02x",getc(stdin)); /* second byte of load address */
    putchar(0x0D); /* followed by a CR */
    for (I = 0; I <= 15; I++) getc(stdin); /* skip next 16 bytes of file */
    for (I = 0; I < bytecount; I++)
    {
        printf("%02x",getc(stdin)); /* first byte of word to monitor */
        printf("%02x",getc(stdin)); /* and second byte, too... */
        putchar(0x0D); /* and a CR for the monitor */
    }
    /* end of WHILE... */
```
putchar(0x1b); /* and an ESC to bail out the monitor */
exit(); /* end of main routine */
}

/* end of main **************************************************************/
J. PIN DESCRIPTION OF IEEE-696/S100 BUS

The 100 pins of the IEEE-696/S100 bus and their functions are described.
<table>
<thead>
<tr>
<th>PIN NO.</th>
<th>SIGNAL &amp; TYPE</th>
<th>ACTIVE LEVEL</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>+8 VOLTS (B)</td>
<td></td>
<td>Instantaneous minimum greater than 7 volts, instantaneous maximum less than 25 volts, average maximum less than 11 volts.</td>
</tr>
<tr>
<td>2</td>
<td>+16 VOLTS (B)</td>
<td></td>
<td>Instantaneous minimum greater than 14.5 volts, instantaneous maximum less than 35 volts, average maximum less than 21.5 volts.</td>
</tr>
<tr>
<td>3</td>
<td>XRDY (S)</td>
<td>H</td>
<td>One or two ready inputs to the current bus master. The bus is ready when both these ready inputs are true. See pin 72.</td>
</tr>
<tr>
<td>4</td>
<td>VI0* (S)</td>
<td>L O.C.</td>
<td>Vectored interrupt line 0.</td>
</tr>
<tr>
<td>5</td>
<td>VI1* (S)</td>
<td>L O.C.</td>
<td>Vectored interrupt line 1.</td>
</tr>
<tr>
<td>6</td>
<td>VI2* (S)</td>
<td>L O.C.</td>
<td>Vectored interrupt line 2.</td>
</tr>
<tr>
<td>7</td>
<td>VI3* (S)</td>
<td>L O.C.</td>
<td>Vectored interrupt line 3.</td>
</tr>
<tr>
<td>8</td>
<td>VI4* (S)</td>
<td>L O.C.</td>
<td>Vectored interrupt line 4.</td>
</tr>
<tr>
<td>9</td>
<td>VI5* (S)</td>
<td>L O.C.</td>
<td>Vectored interrupt line 5.</td>
</tr>
<tr>
<td>10</td>
<td>VI6* (S)</td>
<td>L O.C.</td>
<td>Vectored interrupt line 6.</td>
</tr>
<tr>
<td>11</td>
<td>VI7* (S)</td>
<td>L O.C.</td>
<td>Vectored interrupt line 7.</td>
</tr>
<tr>
<td>12</td>
<td>NMI* (S)</td>
<td>L O.C.</td>
<td>Non-maskable interrupt.</td>
</tr>
<tr>
<td>13</td>
<td>PWRFAIL* (B)</td>
<td>L O.C.</td>
<td>Power fail bus signal. (See Section 2.1.0.1 regarding pseudo open-collector nature.</td>
</tr>
<tr>
<td>14</td>
<td>DMA3* (M)</td>
<td>L O.C.</td>
<td>Temporary master priority bit 3.</td>
</tr>
<tr>
<td>15</td>
<td>A18 (M)</td>
<td>H</td>
<td>Extended address bit 18.</td>
</tr>
<tr>
<td>16</td>
<td>A16 (M)</td>
<td>H</td>
<td>Extended address bit 16.</td>
</tr>
<tr>
<td>17</td>
<td>A17 (M)</td>
<td>H</td>
<td>Extended address bit 17.</td>
</tr>
</tbody>
</table>
| 18      | SDBE* (M)     | L O.C.       | The control signal to disable the 8 status signals.
<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>19</td>
<td>CDSB* (M)</td>
<td>L O.C.</td>
<td>The control signal to disable the 5 control output signals.</td>
</tr>
<tr>
<td>20</td>
<td>GND (B)</td>
<td></td>
<td>Common with pin 100.</td>
</tr>
<tr>
<td>21</td>
<td>NDEF</td>
<td></td>
<td>Not to be defined. Manufacturer must specify any use in detail.</td>
</tr>
<tr>
<td>22</td>
<td>ADSB* (M)</td>
<td>L O.C.</td>
<td>The control signal to disable the 16 address signals.</td>
</tr>
<tr>
<td>23</td>
<td>DDDS (M)</td>
<td>L O.C.</td>
<td>The control signal to disable the 8 data output signals.</td>
</tr>
<tr>
<td>24</td>
<td>f (B)</td>
<td>H</td>
<td>The master timing signal for the bus.</td>
</tr>
<tr>
<td>25</td>
<td>pSTVAL* (M)</td>
<td>L</td>
<td>Status valid strobe.</td>
</tr>
<tr>
<td>26</td>
<td>pHLDA (M)</td>
<td>H</td>
<td>A control signal used in conjunction with HOLD* to coordinate bus master transfer operations.</td>
</tr>
<tr>
<td>27</td>
<td>RFU</td>
<td></td>
<td>Reserved for future use.</td>
</tr>
<tr>
<td>28</td>
<td>RFU</td>
<td></td>
<td>Reserved for future use.</td>
</tr>
<tr>
<td>29</td>
<td>A5 (M)</td>
<td>H</td>
<td>Address bit 5.</td>
</tr>
<tr>
<td>30</td>
<td>A4 (M)</td>
<td>H</td>
<td>Address bit 4.</td>
</tr>
<tr>
<td>31</td>
<td>A3 (M)</td>
<td>H</td>
<td>Address bit 3.</td>
</tr>
<tr>
<td>32</td>
<td>A15 (M)</td>
<td>H</td>
<td>Address bit 15 (most significant for non-extended addressing).</td>
</tr>
<tr>
<td>33</td>
<td>A12 (M)</td>
<td>H</td>
<td>Address bit 12.</td>
</tr>
<tr>
<td>34</td>
<td>A9 (M)</td>
<td>H</td>
<td>Address bit 9.</td>
</tr>
<tr>
<td>35</td>
<td>D01 (M)/DATA1 (M/S)</td>
<td>H</td>
<td>Data out bit 1, bidirectional data bit 1.</td>
</tr>
<tr>
<td>36</td>
<td>D00 (M)/DATA0 (M/S)</td>
<td>H</td>
<td>Data out bit 0, bidirectional data bit 0.</td>
</tr>
<tr>
<td>37</td>
<td>A10 (M)</td>
<td>H</td>
<td>Address bit 10.</td>
</tr>
<tr>
<td>38</td>
<td>D04 (M)/DATA4 (M/S)</td>
<td>H</td>
<td>Data out bit 4, bidirectional data bit 4.</td>
</tr>
<tr>
<td>Pin</td>
<td>Description</td>
<td>Notes</td>
<td></td>
</tr>
<tr>
<td>-----</td>
<td>-------------</td>
<td>-------</td>
<td></td>
</tr>
<tr>
<td>39</td>
<td>D05 (M)/DATA5 (M/S) H</td>
<td>Data out bit 5, bidirectional data bit 5.</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>D06 (M)/DATA5 (M/S) H</td>
<td>Data out bit 6, bidirectional data bit 6.</td>
<td></td>
</tr>
<tr>
<td>41</td>
<td>D12 (S)/DATA10 (M/S) H</td>
<td>Data in bit 2, bidirectional data bit 10.</td>
<td></td>
</tr>
<tr>
<td>42</td>
<td>D13 (S)/DATA11 (M/S) H</td>
<td>Data in bit 3, bidirectional data bit 11.</td>
<td></td>
</tr>
<tr>
<td>43</td>
<td>D17 (S)/DATA15 (M/S) H</td>
<td>Data in bit 7, bidirectional data bit 15.</td>
<td></td>
</tr>
<tr>
<td>44</td>
<td>sM1 (M) H</td>
<td>The status signal which indicates that the current cycle is an opcode fetch.</td>
<td></td>
</tr>
<tr>
<td>45</td>
<td>sOUT (M) H</td>
<td>The status signal identifying the data transfer bus cycle to an output device.</td>
<td></td>
</tr>
<tr>
<td>46</td>
<td>sINP (M) H</td>
<td>The status signal identifying the data transfer bus cycle from an input device.</td>
<td></td>
</tr>
<tr>
<td>47</td>
<td>sMEMR (M) H</td>
<td>The status signal identifying bus cycles which transfer data from memory to a bus master, which are not interrupt acknowledge instruction fetch cycle(s).</td>
<td></td>
</tr>
<tr>
<td>48</td>
<td>sHLTA (M) H</td>
<td>The status signal which acknowledges that a HLT instruction has been executed.</td>
<td></td>
</tr>
<tr>
<td>49</td>
<td>CLOCK (B)</td>
<td>2 MHz (0.5%) 40-60% duty cycle. Not required to be synchronous with any other bus signal.</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>GND (B)</td>
<td>Common with pin 100.</td>
<td></td>
</tr>
<tr>
<td>51</td>
<td>+8 VOLTS (B)</td>
<td>Common with pin 1.</td>
<td></td>
</tr>
<tr>
<td>52</td>
<td>-16 VOLTS (B)</td>
<td>Instantaneous maximum less than -14.5 volts, instantaneous minimum greater than -35 volts, average minimum greater than -21.5 volts.</td>
<td></td>
</tr>
<tr>
<td>53</td>
<td>GND (B)</td>
<td>Common with pin 100.</td>
<td></td>
</tr>
<tr>
<td>No.</td>
<td>Description</td>
<td>Type</td>
<td>Definition</td>
</tr>
<tr>
<td>-----</td>
<td>----------------------</td>
<td>------</td>
<td>---------------------------------------------------------------------------</td>
</tr>
<tr>
<td>54</td>
<td>SLAVE CLR* (B)</td>
<td>L O.C.</td>
<td>A reset signal to reset bus slaves. Must be active with POC* and may also be generated by external means.</td>
</tr>
<tr>
<td>55</td>
<td>DMA0* (M)</td>
<td>L O.C.</td>
<td>Temporary master priority bit 0.</td>
</tr>
<tr>
<td>56</td>
<td>DMA1* (M)</td>
<td>L</td>
<td>Temporary master priority bit 1.</td>
</tr>
<tr>
<td>57</td>
<td>DMA2* (M)</td>
<td>L</td>
<td>Temporary master priority bit 2.</td>
</tr>
<tr>
<td>58</td>
<td>sXTRQ* (M)</td>
<td>L</td>
<td>The status signal which requests 16-bit slaves to assert SIXTN*.</td>
</tr>
<tr>
<td>59</td>
<td>A19 (M)</td>
<td>H</td>
<td>Extended address bit 19.</td>
</tr>
<tr>
<td>60</td>
<td>SIXTN* (S)</td>
<td>L</td>
<td>The signal generated by 16-bit slaves in response to the 16-bit request signal sXTRQ*.</td>
</tr>
<tr>
<td>61</td>
<td>A20 (M)</td>
<td>H</td>
<td>Extended address bit 20.</td>
</tr>
<tr>
<td>62</td>
<td>A21 (M)</td>
<td>H</td>
<td>Extended address bit 21.</td>
</tr>
<tr>
<td>63</td>
<td>A22 (M)</td>
<td>H</td>
<td>Extended address bit 22.</td>
</tr>
<tr>
<td>64</td>
<td>A23 (M)</td>
<td>H</td>
<td>Extended address bit 23.</td>
</tr>
<tr>
<td>65</td>
<td>NDEF</td>
<td></td>
<td>Not to be defined signal.</td>
</tr>
<tr>
<td>66</td>
<td>NDEF</td>
<td></td>
<td>Not to be defined signal.</td>
</tr>
<tr>
<td>67</td>
<td>PHANTOM* (M/S)</td>
<td>L O.C.</td>
<td>A bus signal which disables normal slave devices and enables phantom slaves--primarily used for bootstrapping systems without hardware front panels.</td>
</tr>
<tr>
<td>68</td>
<td>MWRT (B)</td>
<td>H</td>
<td>pWR*--sOUT (logic equation). This signal must follow pWR* by not more than 30 ns. (See note, Section 2.7.5.3)</td>
</tr>
<tr>
<td>69</td>
<td>RFU</td>
<td></td>
<td>Reserved for future use.</td>
</tr>
<tr>
<td>70</td>
<td>GND (B)</td>
<td></td>
<td>Common with pin 100.</td>
</tr>
<tr>
<td>71</td>
<td>RFU</td>
<td></td>
<td>Reserved for future use.</td>
</tr>
<tr>
<td>72</td>
<td>RDY (S)</td>
<td>H O.C.</td>
<td>See comments for pin 3.</td>
</tr>
<tr>
<td>73</td>
<td>INT* (S)</td>
<td>L O.C.</td>
<td>The primary interrupt request bus</td>
</tr>
<tr>
<td></td>
<td>Signal Name</td>
<td>Mode</td>
<td>State</td>
</tr>
<tr>
<td>----</td>
<td>---------------------</td>
<td>------</td>
<td>-------</td>
</tr>
<tr>
<td>74</td>
<td>HOLD* (M)</td>
<td>L</td>
<td>O.C.</td>
</tr>
<tr>
<td>75</td>
<td>RESET* (B)</td>
<td>L</td>
<td>O.C.</td>
</tr>
<tr>
<td>76</td>
<td>pSYNC (M)</td>
<td>H</td>
<td></td>
</tr>
<tr>
<td>77</td>
<td>pWR* (M)</td>
<td>L</td>
<td></td>
</tr>
<tr>
<td>78</td>
<td>pDBIN (M)</td>
<td>H</td>
<td></td>
</tr>
<tr>
<td>79</td>
<td>AO (M)</td>
<td>H</td>
<td></td>
</tr>
<tr>
<td>80</td>
<td>A1 (M)</td>
<td>H</td>
<td></td>
</tr>
<tr>
<td>81</td>
<td>A2 (M)</td>
<td>H</td>
<td></td>
</tr>
<tr>
<td>82</td>
<td>A6 (M)</td>
<td>H</td>
<td></td>
</tr>
<tr>
<td>83</td>
<td>A7 (M)</td>
<td>H</td>
<td></td>
</tr>
<tr>
<td>84</td>
<td>A8 (M)</td>
<td>H</td>
<td></td>
</tr>
<tr>
<td>85</td>
<td>A13 (M)</td>
<td>H</td>
<td></td>
</tr>
<tr>
<td>86</td>
<td>A14 (M)</td>
<td>H</td>
<td></td>
</tr>
<tr>
<td>87</td>
<td>A11 (M)</td>
<td>H</td>
<td></td>
</tr>
<tr>
<td>88</td>
<td>D02 (M)/DATA2 (M/S)</td>
<td>H</td>
<td></td>
</tr>
<tr>
<td>89</td>
<td>D03 (M)/DATA3 (M/S)</td>
<td>H</td>
<td></td>
</tr>
<tr>
<td>90</td>
<td>D07 (M)/DATA7 (M/S)</td>
<td>H</td>
<td></td>
</tr>
<tr>
<td>91</td>
<td>D14 (S)/DATA12 (M/S)</td>
<td>H</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Description</td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>-----------------------------------------------------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>92</td>
<td>D15 (S)/DATA13 (M/S) H</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Data in bit 5 and bidirectional data bit 13.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>93</td>
<td>D16 (S)/DATA14 (M/S) H</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Data in bit 6 and bidirectional data bit 14.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>94</td>
<td>D11 (S)/DATA9 (M/S) H</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Data in bit 1 and bidirectional data bit 9.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>95</td>
<td>D10 (S)/DATA8 (M/S) H</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Data in bit 0 (least significant for 8-bit data) and bidirectional data bit 8.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>96</td>
<td>sINTA (M) H</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>The status signal identifying the bus input cycle(s) that may follow an accepted interrupt request presented on INT*.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>97</td>
<td>sM0* (M) L</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>The status signal identifying a bus cycle which transfers data from a bus master to a slave.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>98</td>
<td>ERROR* (S) L O.C.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>The bus status signal signifying an error condition during present bus cycle.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>99</td>
<td>POC* (B) L</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>The power-on clear signal for all bus devices; when this signal goes low, it must stay low for at least 10 msecs.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>GND (B)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>System ground.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
K. PIN DESCRIPTION OF ZILOG Z-BUS

The full Zilog Z-BUS component interconnection standard is described.
SIGNAL LINES

The Z-BUS consists of a set of common signal lines that interconnect bus components. The signals on these lines can be grouped into four categories, depending on how they are used in transactions and requests.

Primary Signals

These signals provide timing, control, and data transfer for Z-BUS transactions.

ADO-AD15. Address/Data (active High). These multiplexed data and address lines carry I/O addresses, memory addresses, and data during Z-BUS transactions. A Z-BUS may have 8 or 16 bits of data depending on the type of CPU. In the case of an 8-bit Z-BUS, data is transferred on ADO-AD7.

ST0-ST3: Status. (active High). These lines designate the kind of transaction occurring on the bus and certain additional information about the transaction (such as program or data memory access or System versus Normal Mode).

AS. Address Strobe (active Low). The rising edge of AS indicates the beginning of a transaction and that the Address, Status, R/W, and B/W signals are valid.

DS. Data Strobe (active Low). DS provides timing for data movement to or from the bus master.

R/W. Read/Write (Low = write). This signal determines the direction of data transfer for memory or I/O transactions.

B/W. Byte/Word (Low = word). This signal indicates whether a byte or word of data is to be transmitted on a 16-bit bus. This signal is not present on an 8-bit bus.

WAIT. (active Low). A Low on this line indicates that the responding device needs more time to complete a transaction.

RESET. (active Low). A Low on this line resets the CPU and bus users. Peripherals may be reset by RESET or by holding AS and DSLow simultaneously.

CS. Chip Select (active Low). Each peripheral or memory component has a CS line that is decoded from the address and status lines. A Low on this line indicates that the peripheral or memory component is being addressed by a transaction. The Chip Select information is latched on the rising edge of AS.
Bus Request Signals

These signals make bus requests and establish which component should obtain control of the bus.

BUSREQ. Bus Request (active Low). This line is driven by all bus requesters. A Low indicates that a bus requester has or is trying to obtain control of the bus.

BUSACK. Bus Acknowledge (active Low). A Low on this line indicates that the Z-BUS CPU has relinquished control of the bus in response to a bus request.

BAI, BAO. Bus Acknowledge In, Bus Acknowledge Out (active Low). These signals form the bus-request daisy chain.

Interrupt Signals

These signals are used for interrupt requests and for determining which interrupting component is to respond to an acknowledge. To support more than one type of interrupt, the lines carrying these signals can be replicated. (The Z8000 CPU supports three types of interrupts: non-maskable, vectored, and non-vectored.)

INT. Interrupt (active Low). This signal can be driven by any peripheral capable of generating an interrupt. A Low on INT indicates that an interrupt request is being made.

INTACK. Interrupt Acknowledge (active Low). This signal is decoded from the status lines. A Low indicates an interrupt acknowledge transaction is in progress. This signal is latched by the peripheral on the rising edge of AS.

IEI, IEO. Interrupt Enable In, Interrupt Enable Out (active High). These signals form the interrupt daisy chain.

Resource Request Signals

These signals are used for resource requests. To manage more than one resource, the lines carrying these signals can be replicated. (The Z8000 supports one set of resource request lines.)

MMRQ. Multi-Micro Request (active Low). This line is driven by any device that can use the shared resource. A Low indicates that a request for the resource has been made or granted.

MMST. Multi-Micro Status (active Low). This pin allows a device to observe the value of the MMRQ line. An input pin other than MMRQ facilitates the use of line drivers for MMRQ.

MMAI, MMAO. Multi-Micro Acknowledge In, Multi-Micro Acknowledge Out
(active Low). These lines form the resource-request daisy chain.
L. TXMSG: BUS MONITOR MESSAGE RELAY PROGRAM

TXMSG is the Z8000 assembly language program which allows the S100 system to recognize an incoming message, identify it, and relay it to the attached host system.
NMIVECT EQU 0FA16 ... ADDRESS OF THE NMI VECTOR TO SERVICE INCOMING MESSAGES
NMIPSW EQU 0FA14 ... ADDRESS OF STATUS WORD FOR USE DURING MSG HANDLER OPS
REQEN EQU 01130 ... I/O PORT # OF LATCH TO ENABLE THE DATAC TERMINAL ACCESS
REQDIS EQU 01930 ... PORT # TO DISABLE TERMINAL ACCESS OF BUSMON MEMORY
TX_IDHI EQU 01530 ... PORT # OF LATCH HOLDING UPPER BYTE OF MSG ID TAG
TX_IDLO EQU 01030 ... PORT # OF LATCH HOLDING LOWER BYTE OF MSG ID TAG
HALT EQU 07A00 ... (UNIMPLEMENTED OP-CODE UNDER V ASSEMBLER)
NOP EQU 08007 ... ORG 04000

PSTART: JR MAIN ... USE HEX 4000 AS A CONVENIENT STARTING ADDRESS

SETCNT: CLR R1 ... R1 INDEX INTO WEIGHT AND ADDRESS TABLES
NXTWD: LD R5,MATES(R1) ... R5 - POINTER INTO WEIGHT TABLE
       LD R12,STADD(R1) ... R12 - ADDRESS TABLE POINTER
       MULT RR4,R2 ... MULT WORD COUNT * 2 FOR BYTE ADDRESSES
       LD MDC(R1),R5 ... SAVE WORD COUNT
       ADD R12,R5 ... WORD COUNT + START ADDRESS = END ADDRESS
       LD ENDDD(R1),R12 ... SAVE IT...
       INC R1,#2 ... MOVE ON TO NEXT ENTRY
       CP R1,TEMP2 ... AT END OF LIST?
       JR LE,NXTWD ... IF NOT, GO ON TO NEXT ENTRY
       RET

FILL: LD @R8,#OEAE ... FILL DATA AREA WITH NULL DATA
       INC R8,#2 ... ENTER W/ R8= START ADDRESS OF FILL AREA
       CP R8,R5 ... AND R5 = END ADDRESS OF FILL AREA
       JR ULE,FILL ... FILL THE REGION SPECIFIED...
       RET

MAIN: LDA R1,MSG1 ... SETUP AND TRANSMIT PROMPT FOR MODE REQUEST (A/C)
       CLR R2
       LDB RL2,OR1 ... GET BYTE COUNT OF MESSAGE
       LD MSLLEN,R2 ... SAVE IN CONTROL BLK USED BY SC #0
       INC R1,#1 ... ADVANCE R1 TO POINT TO MESSAGE STRING
       LD MSLLOC,R1 ... STORE THAT ADDRESS ON THE CTRL BLK
       LDA R1,PROMPT ... POINT R1 TO THAT BLOCK FOR SC 0
       SC 0 ... DO IT!
       LDA R3,TRANS ... SETUP TO RECEIVE RESPONSE TO MODE PROMPT
       LDA R4,INBUF ... RESPONSE WILL BE SAVED IN INBUF
       LD @R3,R4 ... SET R1 FOR INPUT OPERATION
       SC 0 ... GET THE REPLY
       LD R4,INBUF ... ADJUST RESPONSE TO BCD: GET ASCII "1" OR "2"
       ANDB RH4,#15 ... GET RID OF THE ZONE BITS (03x -> 00x)
       SRL R4,#8
       LD TEMPSTR,R4 ... SAVE THE MODE
       CP R4,#1 ... CHECK TO SEE WHICH MODE
       JR EQ,SKIP ... IF = 1 THEN MODE = 5
       LD MODE,#25 ... IF = 2 THEN MODE = 25
       JR NEXTX
SKIP:   LD  MODE,#5
NEXTX:  LDA  R1,MSG2     ...PREPARE FOR PROMPT OF MULTIPLIER (WORD COUNT)
   CLR  R2
   LDB  RL2,#1       ...SET UP CONTROL BLOCK TO OUTPUT SECOND PROMPT
   LD  MSGLEN,R2
   INC  R1,#1       ... N.B.: RESPONSE TO PROMPT MUST HAVE LEADING
   LD  MSGLOC,R1    ... ZEROES TO KEEP ASCII-TO-HEX ROUTINE HAPPY
   LDA  R1,PROMPT
   SC  0            ...AND PROMPT FOR MULTIPLIER
   LDA  R3,TRANS2    ...PREPARE FOR RESPONSE TO WORD COUNT PROMPT
   LDA  R4,INBUF2
   LD  #R3,R4
   LDA  R1,INPUT2
   SC  0            ...GET RESPONSE (STRING IN INBUF2)
   CLR  R6          ...ADJUST RESPONSE FOR MULTIPLICATION
   CLR  R3          ... R4, R3, AND R6 ARE USED IN THE ASCII TO
   CLR  R4          ... HEX NUMBER CONVERSION ROUTINE
   LO  R6,#2
   LDB  RH2,INBUF2(R3)...GET FIRST ASCII DIGIT
HELLO:  INC  R3,#1      ...ADVANCE POINTER TO NEXT DIGIT IN STRING
   ANDB  RH2,#15     ...GET RID OF ZONE BITS ON THE ASCII DIGIT
   SRL  R2,#8
   ADD  R4,R2        ...SAVE THE CURRENT DIGIT IN R4
   CP  R3,#3         ...CHECK TO SEE IF WE'VE PROCESSED 3 DIGITS YET
   JR  EQ,FINI       ...IF SO, WE'RE DONE!
   LD  R1,R4         ...OTHERWISE
   MULT  RR0,#0A     ...MULTIPLY THE CURRENT DIGIT BY 10 (HEX)
   LD  R4,R1         ...AND SAVE IT IN OUR ACCUMULATOR
   LDB  RH2,INBUF2(R3)...GET NEXT DIGIT FROM STRING
   JR  HELLO         ...AND PROCESS IT
FINI:   DEC  R4,#1      ...ADJUST WORD COUNT VALUE FROM N TO PROVIDE 0 - (N-1) RANGE
   LD  TEMPSTR(R6),R4...AND SAVE IT
   LD  R1,MODE       ...RECALL MODE # (1 FOR A, 2 FOR C)
   SUB  R1,#1        ...ADJUST FOR 0 OR 1 RANGE
   LD  R2,#2
   MULT  RR0,R2      ...NOW RANGE FOR 0 OR 2
   LD  TEMP2,R1      ...AND SAVE IT
   LD  R2,R4         ...WORD/MODE MULTIPLIER (0/2)
   CALLR  SETCNT     ...CALL SUBROUTINE TO SET WORD COUNT
   LD  R8,#08000     ...START ADDRESS OF MESSAGE DEPOSIT AREA
   LD  R5,#08500     ...END ADDRESS OF MESSAGE AREA
   CALR  FILL        ...FILL DATA AREA
   LD  NMIPSW,#04000  ...NMI HANDLER
   LD  NMIVECT,#GETDAT ...VECTOR FOR INTERRUPT
NXTMSG: OUT  REQEN,R5   ...ENABLE NMI
STALL:  OUT  REQEN,R5   ...THIS SEEMS TO HELP AVOID LOCKOUTS
   JR  STALL        ...DO NOTHING BUT WAIT FOR A MESSAGE
GETDAT: INB  RH5,TX_IDHI ...INPUT HI & LO BYTE OF ID TAG
   INB  RL5,TX_IDLO
   LD  R6,R5
   LD  IDTAG,R6     ... SAVE ID TAG IN MEMORY
LD R5,ENDAD(R6)  ... LOAD END ADDRESS

LOOP:
CP @R5,#OEAEA  ... COMPARE END ADD WITH (OEAEA)
JR EQ,LOOP  ... CHECK FOR CHANGE IN LAST DATUM
OUT REQDIS,R5  ... DISABLE NMI PROCESSING WHEN VALUE CHANGES
INC R15,#6
LD R8,STADD(R6)  ... LOAD BEGIN ADDRESS
LD R9,WDCT(R6)  ... LOAD NORD COUNT
INC R9,#2  ... INCREMENT WORD CNT TO INC LAST DATA
LD BEGAD,R8  ... PLACE BEGIN ADD FOR DATA TRANSMISSION
LD ENDAO,R9  ... PLACE WDCT IN END ADD FOR TRANS
LD R1,#SETUP  ... PREPARE FOR DATA DUMP
LDM REG_SAVE,R0,15  ... SAVE REGISTERS BEFORE SC 0
SC 0  ... SEND DATA TO WAITING CONSOLE SYSTEM
LDM R0,REG_SAVE,15  ... RESTORE REGISTERS AFTER SC 0
LD R11,R9  ... BACK UP TO OFFSET OF LAST MESSAGE WORD
DEC R11,#2  ... GET ADDRESS OF THAT WORD
ADD R11,#08000  ... BY ADDING MSG AREA BASE ADDRESS
NWDT:
LD @R11,#OEAEA  ... AND NULL THEM OUT
ADD R11,#0100  ... MOVE TO NEXT MSG SAVE AREA
CP R11,#08500  ... DONE THEM ALL?
JR ULT,NEXT  ... IF NOT, DO SO...
LD NMIPSW,#04000  ... RESTORE CONTEXT FOR NMI HANDLER
LD NMI_VECTOR,#GETDATA  ... AS WELL AS NMI VECTOR FOR INTERRUPT PROCESSING
JR NXTMSG  ... WAIT FOR NEXT MESSAGE

IDTAG:  WORD 0  ... TEMP STORAGE FOR RECONSTRUCTED ID TAG
REG_SAVE:  WORD [32]  ... SAVE AREA FOR REGISTER FILE DUMP DURING SC #0
SETUP:  WORD 0200
TRANS:  WORD 0
INPUT2:  WORD 0100
TRANS2:  WORD 0
INBUF:  DEFS 80
INBUF2:  DEFS 80
TEMPSTR:  WORD 0[10]
TEMP2:  WORD 0
ORG 07000  ... DATA DEF TABLE FOR MODE A SIMULATION
STADD:  WORD 08000 08100 08200 08300 08400  ... START ADDRESSES

WORD  08000 08100 08200 08300 08400
WORD  08000 08100 08200 08300 08400
WORD  08000 08100 08200 08300 08400
WDCT:  WORD  0[25]  ...NUMBER OF WORDS APPEARING UNDER EACH LABEL/FRAME
ENDADD:  WORD  0[25]  ...LAST ADDRESS MONITORED FOR EOTransfer
MATES:  WORD  2[25]  ...NORMALIZED WEIGHTS FOR COUNT CALC
MODE:  WORD  0  ... 5 = MODE A, 25 = MODE C; USED IN TABLE GENERATION
END:
/IMAGE TXMSG.HEX PSTART END E=PSTART
//
M. COMSAVE: MESSAGE RECEIVER/STORAGE PROGRAM

COMSAVE is the 8086 assembly language program for the IBM PC which monitors the serial channel which connects the PC to the S100 systems, captures and saves data appearing on that channel.
COMSAVE

1. Opens file using the DOS 2.xx or 3.xx file handle method.

2. Reads data from COM1: until one of the following conditions is met:
   a. user strikes key on PC keyboard.
   b. 64K bytes read from COM1:.

   At this time any further data read from COM1: is ignored.

3. When condition 2.a. or 2.b. is data is copied into the opened file.

This program is derived from routines appearing in the Waite series on 808x assembly language programming. Most of the procedures have been modified, drastically in a few cases, to suit this particular application.

Written:  R. N. Turner
Avionics Engineering Center
Ohio University
July 1986

;===============================================
; cr   equ       0Dh
; lf   equ       0Ah
;===============================================

dseg segment ;data segment
nambuff db    49 ;max bytes allowed in filespec.
db   ?         ;number of bytes entered
db   50 dup(?) ;ASCIIZ filespec buffer
prompt db     cr,lf,'Enter filespec: $'
crlf db       cr,lf,'$'
escape db     cr,lf,'<esc>',cr,lf,'$'
com_config db 11110011b ;9600 baud ,N parity, 1 stop, 8 bit wd
bell db       07h,'$' ;beeper
port_no dw    0000h
file_handle dw ? ;storage for the file handle
max_bytes dw   OFFFFh ;maximum number of bytes to save
intro db      cr,lf,lf,lf,'COMSAVE',
db            cr,lf,lf,'Program to store up to 64 Kbytes of data'
'from COM1: into a disk file.', cr, lf
'will be overwritten.', cr, lf,'$
File cannot previously exist or else it '
'File creation error.$'
File close error.$'
File write error.$'
'Memory full. $'
'User intervention. $'
'Saving captured data.$'
'Finished.$'

*************************************************************************

*************************************************************************

*************************************************************************

*************************************************************************

*************************************************************************

*************************************************************************

*************************************************************************
mov ds,ax
;call big_guy ;this does most everything
;ret

main endp

---------------------------------------------

big_guy proc near

;this procedure implements all of the following functions:

1. set up COM1: serial port for 9600 baud, no parity, 8 data bits, 1 stop bit operation.

2. request a name for a disk file to be used to store data obtained from the outside world through COM1:. The max. length of the file specification is 50 characters. This includes drive and path name and everything.

3. capture data from COM1: and store it temporarily in a memory area defined by the extra segment register, es:.

4. halt capturing data after either of the following conditions:
   a. 64 Kbytes of data have been captured from COM1:.
   b. the user has struck the ESC key on the PC keyboard.

5. the data is copied from the extra segment into the disk file and the file is closed.

The program may terminate upon encountering the following conditions:

1. error opening the disk file. Opening an existing file is probably not desired but will not be detected by this program, at least not by this version.
2. Error writing to the disk file which, for example, can occur if the disk fills up.
3. Error closing the disk file. This shouldn't happen unless you're using a hard disk and you already have something on the order of 512 files in the directory you tried to save the file in.

;read in filespec of file to be opened

newfile:

mov dx,offset intro ;welcome user
call type_string
mov dx,offset prompt ;prompt for output filespec
call type_string

mov dx,offset nambuff ;address of buffer
mov ah,0Ah ;buffered keyboard input function
int 21h ;call DOS

mov dx,offset crlf ;go to newline
call type_string

; put a null at the end of the filespec to make it an ASCIIZ string
mov bl,nambuff+1 ;get number of bytes read
mov bh,00h ; into bx register
mov [nambuff + bx + 2],00h ;make the ASCIIZ string

; set ds:dx to point to the ASCIIZ string
mov dx,offset nambuff + 2 ;address of filespec

; create the output file

call fcreate
mov file_handle,ax ;put file handle in safe place
jnc open_ok ;no error occurred, continue
mov dx,offset open_error ;error on file creation
jmp error ;go display error message

open_ok:

; initialize COM1:

mov al,com_config
mov dx,port_no
call cominit ;set port parms
call comon ;enable com: port

; set up pointer into ES: segment where we're going to store bytes collected
; from COM1:

mov di,0000h ;index into array of msg bytes

nxtbyt:

; check for input from COM1:. If found store it at es:di.

mov dx,port_no
call cominck ;look for com: input
jz qkey ;nothing sent, nothing to store

assume es:eseg
mov bx,eseg
; make sure it ends up at 3000:
mov es, bx
mov es:[di], al

; save the byte read from COM1:

; check to see if we've written 64K bytes. If so we have to quit reading
; and storing data.
inc di
cmp dl, max_bytes
je seg_end

; move on the the next save location
inc di
jmp click

; are we pointing to last location in segment
je seg_end

; yes, sorry... gotta go
call click

; make a click on speaker to mark data read

; check for a keystroke from the PC. If we find one look to see if it's
; the ESC key. If so terminate COM1: data collection.

push ax
mov ah, 01h
int 16h
jz nokey1
mov ah, 00h
int 16h
jz nokey1

cmp al, 18h
jne nokey1

; did we read an escape key?

; no, continue.

; restore ax after keyboard check.
mov dx, offset escape
call type_string
jmp user_end

; yes, let's copy data from memory to a file!

nokey1:
pop ax
jmp nxxbyte

; restore ax after keyboard check.

user_end:
mov dx, offset user_stop
invoke type_string
jmp data_copy

; display message indicating that we've ended because of user intervention.
seg_end:
mov dx, offset seg_stop
invoke type_string
jmp data_copy

; display message indicating that we've ended because storage segment is full.
data_copy:
mov dx, offset saving
invoke type_string

; save ds because we have to change it to point to the extra segment area
; for the file write.

push ds

; set up for write via file handle call
mov bx, file_handle
mov ah, 0h
int 16h

mov    ax, es      ; we get segment pointed to by es: and
mov    ds, ax      ; set ds: to point to that area
mov    dx, 0000h
mov    cx, di      ; write as many characters as we stored
mov    ah, 40h     ; write to file function
int     21h        ; call DOS
pop     ds         ; get original ds: register contents
jnc     fini
mov    dx, offset write_error ; get set to dump out write error msg
jmp     error

fini:    
call    fclose
jnc     cont2
mov    dx, offset close_error ; point to file close error msg
jmp     error

cont2:   
mov    dx, offset done_msg ; point to termination message
jmp     error ; not really this time, just convenient

error:   
call    type_string ; display error message.

theend:  
ret      ; exit to DOS.

big_guy endp

;-----------------------------------------------------------------------------

;-----------------------------------------------------------------------------

binidec proc near

; procedure to display bx register contents in decimal on the screen

mov    cx, 10000d ; divide by 10000
call    dec_div
mov    cx, 1000d ; divide by 1000
call    dec_div
mov    cx, 100d ; divide by 100
call    dec_div
mov    cx, 10d ; divide by 10
call    dec_div
mov    cx, 1d ; divide by 1
call    dec_div

ret      ; yes, return to caller

binidec endp

;-----------------------------------------------------------------------------

;-----------------------------------------------------------------------------

dec_div proc near

; procedure to divide the number in bx by the number in cx. the quotient
is displayed on the screen. the numerator is in ax+dx, denom in cx.

: mov ax,bx ;number in high half
mov dx,0000h ;zero out low half
div cx ;divide by cx
mov bx,dx ;remainder in bx
mov dl,al ;get quotient in dl

: print quotient on screen
 add dl,30h ;make quotient an ASCII digit
mov ah,02h ;DOS print function
int 21h ;call DOS
ret

def_div endp

-------------------------------------------------------------

: COMON

;Function: This routine turns on the handshaking signals DTR (line 20) and
; RTS (line 4) on the specified communications line.

;Input: Upon entry DX contains the unit number (0 for COM1: or 1 for COM2:).

;Output: Just to the communications line.

;Registers used: No registers are modified. DX is used for input.

;Segments referenced: During the routine the system data segment is
; referenced.

;Routines called: None

;Special notes: None

comon proc near
push ds
push dx
push si

mov si,dx ;look up address of comm line
add si,si ;double index into table
mov dx,0040h ;segment of BIOS data storage area
mov ds,dx ;set data segment to this table
mov dx,[si] ;now get the table
add dx,04h ;modem control register
mov al,03h ;set DTR and RTS
out dx,al ;send out the control byte
Function: This routine checks, but does not wait for input from one of the two serial communications lines.

Input: Upon entry DX contains the unit number (0 for COM1: or 1 for COM2:).

During the routine, input is from the specified communications line.

Output: If a byte is available, the routine returns the flag condition NZ and the byte is in AL, but if no byte is available then the routine returns with the flag condition I, and AL is meaningless.

Registers used: AH is modified. DX is used for input and AL is used for output.

Segments referenced: During the routine the system data segment is referenced.

Routines called: None

Special notes: None

; routines called: none

; special notes: none

; function: this routine checks, but does not wait for input from one of the two serial communications lines.

; input: upon entry dx contains the unit number (0 for com1: or 1 for com2:).

; during the routine, input is from the specified communications line.

; output: if a byte is available, the routine returns the flag condition nz and the byte is in al, but if no byte is available then the routine returns with the flag condition i, and al is meaningless.

; registers used: ah is modified. dx is used for input and al is used for output.

; segments referenced: during the routine the system data segment is referenced.

; routines called: none

; special notes: none

cominck proc near

push ds
push dx
push si

mov si, dx ; lookup address of com line
add si, si ; double index into table
mov dx, 0040h ; segment of bios data table
mov ds, dx ; set data segment to this table
mov dx, [si] ; now get table data
add dx, 05h ; line status
in al, dx ; get status ???

; receive buffer full?

jz cominckexit

mov dx, [si] ; data register
in al, dx ; get data ???

cominckexit:
pop si           ; restore registers
pop dx
pop ds
ret

; cominck endp
;
;---------------------------------------------------------------------
;
; CLICK
;
;Function: This procedure makes a short click on the speaker.
;
;Input: None
;
;Output: To the speaker and timer only.
;
;Registers used: No registers are modified.
;
;Segments referenced: None
;
;Routines called: None
;
;Special notes: None.
;
;click proc near
;
;   push ax           ; save registers
;   turn speaker and timer on
;   in   al,61h      ; get contents of system port B
; or    al,3         ; turn timer and speaker on
; out   61h,al       ; send out new values to port B
; turnoff timer 2 and speaker
; and   al,11111100b  ; turn off timer and speaker
; out   61h,al       ; do it
; pop   ax           ; restore registers
; ret

; click   endp
;
;---------------------------------------------------------------------

; COMINIT
;
;Function: This routine initializes one of the two communications lines.
;
;Input: Upon entry AL contains the initialization byte as followes:
;
;   7 6 5 4 3 2 1 0
;   --baud rate-- --parity-- --stop bits-- --word length--
;   000 = 110 00 = none 0 = 1 10 = 7 bits
; 001 = 150  01 = odd    1 = 2    11 = 8 bits
; 010 = 300  10 = none
; 011 = 600  11 = even
; 100 = 1200
; 101 = 2400
; 110 = 4800
; 111 = 9600
; DX contains the unit number, 0 for COM1: or 1 for COM2:

; Output: Output is sent to the hardware controlling the specified communications line.

; Registers used: No registers are modified.

; Segments referenced: None

; Routines called: BIOS interrupt 14h (RS-232 I/O) is used.

; Special notes: None

cominit proc near

mov ah,0 ; initialize comm line function
int 14h ; BIOS RS-232 call
ret

cominit endp

;--------------------------------------------

fcreate proc near

; Creates a file using the 'file handle' method.

; Upon entry, dx points to the ASCIIZ filespec of the file to be created.
; ds is assumed to be correctly set.
; ax contains the file handle when done.
; all other registers are not affected.

push cx ; save registers
push dx

mov ah,3Ch ; create file function
mov cx,0000h ; 'normal' file attribute
int 21h ; call DOS

pop dx
pop cx

ret
\texttt{fclose proc near}
\texttt{; closes a file opened or created using the file handle method of access.}
\begin{verbatim}
mov ah,3Eh ; close file handle function
int 21h ; call DOS
ret
\end{verbatim}

\texttt{fclose endp}

\texttt{type_string proc near}
\texttt{; uses DOS interrupt 21h display string function to dump the string pointed to by ds:dx.}
\begin{verbatim}
push ax ; save registers used
mov ah,09h ; display string function
int 21h ; call DOS
pop ax ; restore registers used
ret
\end{verbatim}

\texttt{type_string endp}

\texttt{cseg ends}
\begin{verbatim}
end start ; end assembly
\end{verbatim}
N. DSCAN: MESSAGE TABULATOR/PREFORMATTER PROGRAM

DSCAN is a C language program which converts the hexadecimal data from the Bus Monitor to ASCII for use by data reduction programs.
DSCAN: program which accepts hexadecimal data output from the S100 system captured by the COMSAVE program, tabulates the values and produces an ASCII report file of those values. That report is to be imported into a Lotus 1-2-3 worksheet for subsequent timing analysis. DSCAN produces, for each message received by the S100 system, a record which consists of the frame in which the message occurred, the ID tag of the system which "sent" the message, and the number of words which appeared in that message. This last piece of information might seem extraneous in light of the operator having entered that value, but superfluous data was found in the COMSAVE data files (we never found the source of this spurious data) and it was desired to check on the integrity of the messages. A typical output line might resemble this:

```
4 0023 18
```

where "4" is the system ID, "0023" is the frame number of the message, and "18" is the number of words appearing in that message. The 1-2-3 worksheet accepted these values and produced intermessage time values, average intermessage times, and other statistics about the COMSAVE data file.

*/

int eoflag; /* a flag value is used because somewhat extensive housekeeping is necessary when EOF is encountered on the data file. An integer value is used because C doesn't have a LOGICAL data type as such */

main(argc,argv)
int argc;
char *argv[];
{
    /* beginning of MAIN */
    FILE *fp,*fopen(); /* the standard C files for command line argument access... */

    int dcount, ref_datum[5], ref_count[5], datum[100], datum_count,
        lstid, maxlim, i, j, onlist, max, max_at, x, y,
        lstfrm, offcnt, lst_word, this_step, lst_step;

    /* initialization of various flags for the first data scan */
    lstfrm = -1; lstid = 0; offcnt = 0; lst_word=0; lst_step=0;

    /* open the data file specified on the command line */
    if ((fp = fopen(*++argv,"rb")) == NULL)
    {
        fprintf(stderr,"dscan: can't open file %s
","argv);
        exit(1);
    }
    else /* which means the open was successful */
    {
dcount = atoi(*++argv); /* convert the words-per-message value from ASCII to internal integer */
if (dcount == 0)
{
    fprintf(stderr, "Valid count value must be specified \n");
    exit(1);
}

maxlim = (int)(1.50*(float)dcount); /* dynamically set filter constant for bad data */
if (maxlim > 100) maxlim = 100; /* based on number of words in message */

fprintf(stderr, "Slide of %d allowed on msg sync errors...\n", maxlim);

/* this header enables the casual user to tell which columns of numbers are what */
printf("\n Station: Frame: Word Count:\n\n", dcount);

eoflag = 0; /* to get things going */

for (;;) /* the proverbial infinite loop, used because of the need to manipulate the */
{ /* end of file condition for housekeeping and other clean ups */
for (datum_count = 0; datum_count < dcount; datum_count++) /* read DCOUNT words from the file */
{
    datum[datum_count] = getdatum(fp); /* save a msg from the station */
    if (eoflag == 1)
    {
        if (datum_count == 1) gete finito;
        dcount = datum_count + 1;
        goto lst_msg; /* use GOrO cause u can't BREAK out two levels deep */
    } /* endif */
} /* endfor */

lst_msg: /* GOTO here when the data file is empty...*/
ref_datum[0] = 0; /* these are used to tabulate the 5 most common word */
ref_datum[1] = 0; /* values from the message, necesssary because of data */
ref_datum[2] = 0; /* contamination during the serrial Rx/Tx procedure */
ref_datum[3] = 0;
ref_datum[4] = 0;
ref_count[0] = 0;
ref_count[1] = 0;
ref_count[2] = 0;
ref_count[3] = 0;
ref_count[4] = 0;

/*------- find the 5 (or fewer) different words in the message -------*/
x = 0; /* current # of ref_data entries */
for (i = 0; i < dcount; i++) /* run through DCOUNT words in the message */
{
    /* and find the most common ones */
    y = x;
onlist = 0;
    for (j = 0; j < x; j++) /* check and see if current datum is in lst */
if (datum[i] == ref_datum[j]) onlist = 1;

if (onlist == 0) /* if not currently on the data list, enter it */
/* EAEA and EEEE are the known context-insensitive words */
/* the current frame is extracted and checked against the next frame, which helps */
/* find some of these boffo words */
{
    if (
        (datum[i] != 0xEAEA) &&
        (datum[i] != 0xEEEE) &&
        ((datum[i] & 0xFFFF) > lstfrm) ||
        ((datum[i] & 0xFFFF) == lstfrm) &&
        (datum[i] >> 12) > lstid)
    )
    /* count only fresh words */
    ref_datum[x] = datum[i]; /* EEEE and EAEA are null data */
    y++;
    /* and count the number of unique table entries */
    if (y == 5)
    {
        y = 4;
        goto quitseek; /* once again, you can't BREAK out more than 2 {}s deep */
    } /* endif-y */
} /* endif-onlist */

x = y; /* done so we don't change counter x in the middle of a FOR loop */
} /* end of for-loop (i) to seek out 5 diff words */

quitseek:

 BusinessException: 

/*------------- tabulate occurrences of the 5 or fewer words found -------*/
for (i = 0; i < x; i++)
    for (j = 0; j < dcount; j++)
        if (ref_datum[i] == datum[j]) ref_count[i]++;

/* ---------- locate most frequently occurring word and its assoc count ---*/
max = 0;
max_at = 0;

for (i = 0; i < x; i++)
    if (ref_count[i] > max)
    {
        max_at = i;
        max = ref_count[i];
    }
--- the output line in the report file consists of:

```c
    datum[max_at] and max
    |          | ------ % of occurrences this datum
    |          | ------ most freq occurring word in msg
```

```c
if (eoflag == 1) break; /* done with our infinite read */

print_count_line(datum[max_at], max); /* output the current record */
lstfrm = datum[max_at] & 0x0FFF; /* extract the latest frame % for corruption checks */
lstid = datum[max_at] >> 12; /* and the same for the latest station % */

if (max < dcount) /* this section checks the current frame and station %s against */
{
    /* the previous values, is there's a big discrepancy, we missed a */
    offcnt++; /* bad word */
    if (offcnt > maxlim)
    {
        fprintf(stderr,
            "\007\nPossible msg sync error near %04d!!\n",
            (datum[max_at] & 0x0FFF)-(offcnt*lst_step)); /* long message... */
        offcnt = 0; /* reset for next potential disaster */
    } /* endif on offcnt */
    else offcnt = 0; /* endelse on max */
}

this_step = (datum[max_at] & 0x0FFF) - (lst_word & 0x0FFF);
if (this_step > (lst_step + 5)) goto daterr; /* no more than 5 frame skip allowed */
/* estimated BusMon performance for this test should not have more than 5 frames */
/* skipped, if this happens, assume that we have a major data bumble... */

lst_word = datum[max_at];
lst_step = this_step;
}
/* end of while */

finito: fclose(fp);
exit(0);

daterr: fprintf(stderr,"\007\nPossible Data Sync error in data file %d:%s\n",
    dcount, *argv);
fclose(fp);
exit(10);

} /* end of main */

="/---------------------------------------------*/

print_count_line(a,b)
int a, b:
/* formatted outputter for the records in the report file */
{
printf("%1d %04d %3d
",(a >> 12),(a & 0xFFF),b);
}

getdatul(fp)
FILE *fp;
{
int c,d,j;
/*
this function assembles the 16-bit word values byte-serially. this is necessary
because the C EOF recognition function relies on CHAR values of 0-255 and a -1
value (FFFF hex) is a valid unsigned 16-bit value. So we read the words 8 bits
at a time, put them together and let C detect EOF for us
*/
gethibyt: if (c = getc(fp)) == EOF) /* get high order byte of data word */
{
    eoflag = 1;
    goto done;
}
/* once again we filter out known boffo data */
if ((c == OxO) || (c == Ox64) || (c == Ox47)) goto gethibyt: /* disallow spurious stuff */
/* low bytes, as that would really mess things up... */

getlobyt: if ((d = getc(fp)) == EOF) /* get the low order byte */
{
    eoflag = 1;
    goto done;
}

j = (c << 8) | d; /* assemble the two bytes into an unsigned 16-bit word */
done: {}
return(j);
0. MPU 8000 II SCHEMATIC

The circuit diagram for the Ithaca Intersystems MPU 8000 II Z8002-based CPU card is presented.
P. TRUMP CARD SCHEMATIC

The circuit diagram for the Sweet Microsystems Trump Card Z8001-based SS-62 coprocessor card is presented.