intel®

APPLICATION NOTE

AP-274

May 1986

Ethernet/Cheapernet LAN Design

Kiyoshi Nishide
APPLICATIONS ENGINEER

Order Number: 292010-001

Additional copies of this manual or other Intel literature may be obtained from:

Intel Literature Inquiries
SC6-58
P.O. Box 58065
Santa Clara, CA 95052-8065

ETHERNET/CHEAPERNET LAN DESIGN

CONTENTS

PAGE

CONTENTS

PREFACE

Intel's three VLSI chip set, the 82586, 82501, and 82502, is a complete solution for IEEE 802.3 10M bps LAN standards—10BASE5 (Ethernet) and 10BASE2 (Cheapernet). The 82586 is an intelligent peripheral which completely manages the processes of transmitting and receiving frames over a network under the CSMS/CD protocol. The 82586 with its on-chip four DMA channels offloads the host CPU of the tasks related to managing communication activities. The chip, for example, does not depend on the host CPU for time critical functions, such as transmissions/retransmissions and receptions of frames. The 82501 is a 10 MHz serial interface chip specially designed for the 82586. The primary function of the 82501 is to perform Manchester encoding/decoding, provide 10 MHz transmit and receive clocks to the 82586, and drive the transceiver (AUI) cable in Ethernet applications. In addition, the 82501 provides a loopback function and on-chip watchdog timer. The 82502 is a CMOS transceiver chip. The 82502 is the chip which actually drives the coaxial cable used for Ethernet or Cheapernet.

This Ap Note presents a design example of a simple but general Ethernet/Cheapernet board based on the three chip set. The board is called LANHIB (LAN High Integration Board) and uses an 80186 microprocessor as the host CPU. The LANHIB is an independent single board computer and requires only a power supply and ASCII terminal. Demo software, called TSMS (Traffic Simulator and Monitor Station) is also included in this Ap Note. The TSMS program is a network debugger and exercise tool used to exercise the 82586. In addition, flowcharts for troubleshooting are provided in order to minimize debugging time of the LANHIB board.

1.0 INTRODUCTION

A brief overview of the CSMA/CD protocol is described in Section 2. Ethernet and Cheapernet are also compared in this section.

Section 3 discusses hardware of the LANHIB in detail. This section should be helpful not only to understand the LANHIB, but also to learn in general how a system based on the three chip set can be put together. Since the 82502 involves analog circuitry, an explanation on proper layout is provided.

Demo software is presented in Section 4.0. It covers EPROM programming procedures and three sample sessions. Step by step operations at a terminal are illustrated in the figures.

Section 5 describes LANHIB troubleshooting procedures. Flowcharts are used to guide troubleshooting.

Complete LANHIB schematics and parts list are found in Appendix A. If a LANHIB is to be built, the schematics and Section 5 can be submitted to an available wire wrap facility. In parallel to board construction, Sections 3 and 4 can be studied. A factory wire wrap board for the LANHIB is offered at a discount price by Augat Corporation. Please return the enclosed card for more information.

Listing of the TSMS program and LANHIB Initialization Routine are in Appendix B. The source codes and related files are available on a diskette by returning the card enclosed in this design kit or through Insite (Intel's Software Index and Technology Exchange Library).

2.0 ETHERNET/CHEAPERNET OVERVIEW

2.1 CSMA/CD

Carrier Sense Multiple Access with Collision Detection (CSMA/CD) is a simple and efficient means of determining how a station transmits information over common medium that is shared with other stations. CSMA/CD is the access method defined by the IEEE 802.3 standard.

Carrier Sense (CS) means that any station wishing to transmit "listens" first. When the channel is busy (i.e., some other station is transmitting) the station waits (defers) until the channel is clear before transmitting ("listen before talk").

Multiple Access (MA) means that any stations wishing to transmit can do so. No central controller is needed to decide who is able to transmit and in what order.

Collision Detection (CD) means that when the channel is idle (no other station is transmitting) a station can start transmitting. It is, however, possible for two or more stations to start transmitting simultaneously causing a "collision". In the event of a collision, the transmitting stations will continue transmitting for a fixed time to ensure that all transmitting stations detect the collision. This is known as jamming. After the jam, the stations stop transmitting and wait a random period of time before retrying. The range of random wait times increases with the number of successive collisions such that collisions can be resolved even if a large number of stations are colliding.

There are three significant advantages to the CSMA/CD protocol. The first and foremost is that CSMA/CD is a proven technology. One CSMA/CD network, Ethernet, has been used by Xerox since 1975. Ethernet is so well understood and accepted that IEEE adopted it (with minor changes) as the IEEE 802.3 10Base5 (10 Mbps, Baseband, 500 meters per segment) standard. Reliability is the second advantage to the 802.3 protocol. This media access method enables the network to operate without central control or switching. Thus, if a single station malfunctions, the rest of the network can continue operation. Finally, since CSMA/CD networks are passive and distributed in nature, they allow for easy expansion. New nodes can be added at any time without reinitializing the entire network.

2.2 Ethernet and Cheapernet

The IEEE 802.3 Type 10BASE5 standard (Ethernet) has gained wide acceptance by both large and small corporations as a high speed (10 Mbps) Local Area Network. The Ethernet channel is a low noise, shielded 50Ω coaxial cable over which information is transmitted at 10 million bits per second. Each segment of cable can be up to 500 meters in length and can be connected to longer network lengths using repeaters. Repeaters regenerate the signal from one cable segment onto another. At each end of a cable segment a terminator is attached. This passive device provides the proper electrical termination to eliminate reflections. The transceiver transmits and receives signals on the coaxial cable. In addition, it isolates the node from the channel so that a failure within the node will not affect the rest of the network. The transceiver is also responsible for detecting collisions—simultaneous transmissions by two or more stations. Ethernet transceivers are connected to the network coaxial cable using a simple tap, and to the station it serves via the transceiver cable which can be

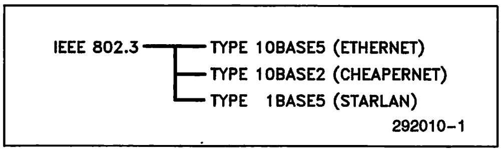


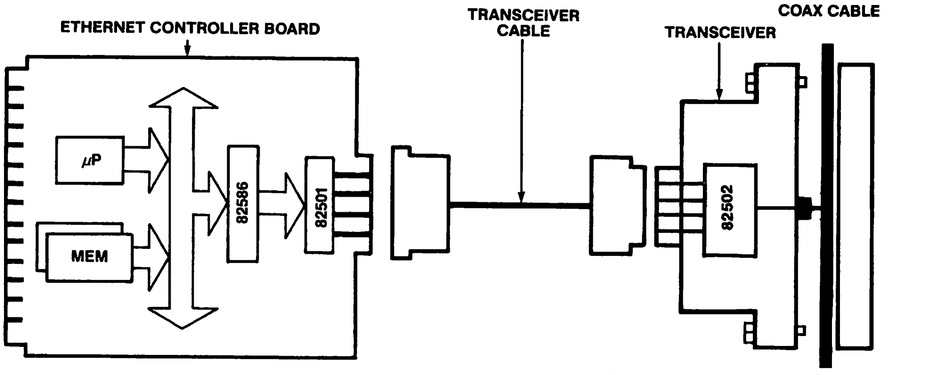
Figure 1. Different Implementations of IEEE 802.3 (Note: "10BASE5", for example, implies 10 Mbps, Baseband, and 500 meters span.)

up to 50 meters in length. The transceiver cable is made of four individually shielded twisted pairs of wires. An Ethernet interface at a computer (DTE), which includes a serial interface and data link controller, provides the connection to the user or server station. It also performs frame manipulation, addressing, detecting transmission errors, network link management, and encoding and decoding of the data to and from the transceiver.

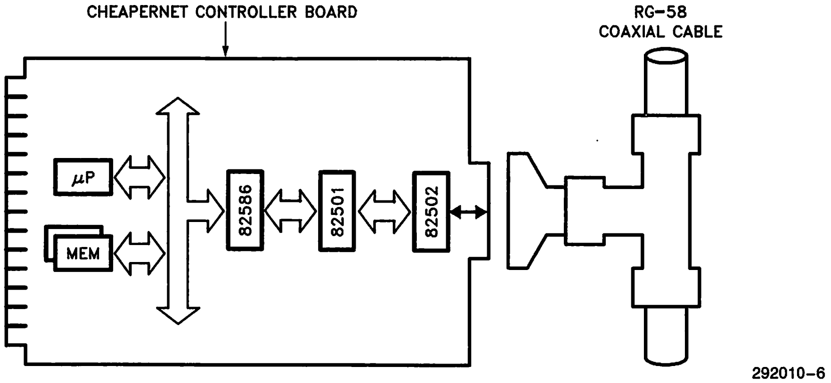
The IEEE 802.3 Type 10BASE2 (Cheapernet) has the same functional and electrical specifications as Type 10BASE5 (Ethernet) with only two exceptions in physical (or rather mechanical) characteristics. Cheapernet is as shown in Figure 1 just a different implementation of the IEEE standard. Ethernet and Cheapernet are both 10 million bits/second CSMA/CD LANs and use the identical network parameters, such as slot time = 51.2 μs. Ethernet and Cheapernet can, therefore, be built by the same VLSI components with the same software (Figure 2).

The two physical differences attribute to the cost reduction purpose of Cheapernet—cheaper implementation of Ethernet. First, the cable used in Cheapernet may be a lower cost 50Ω coaxial cable than the one for Ethernet. The most common coaxial cable for Cheapernet is RG58 which cost about $0.15/ft. A typical Ethernet cable costs about $0.83/ft.

Second, the transceiver is integrated into the DTE in Cheapernet. The coaxial cable physically comes to the DTE, connects to the transceiver within the DTE, and goes to the next DTE (see Figure 3). The kind of connector used at the DTE is an off-shelf BNC "T" connector. Topology is, therefore, a simple daisy chaining. This cabling scheme contributes to further cost reduction due to omission of the Transceiver (AUI) Cable, cheaper connectors, and easier installation. The Ethernet transceiver cable costs about $1.49/ft. More flexible thin coaxial cables and familiar BNC "T" connectors are making Cheapernet a user installable Ethernet compatible network.

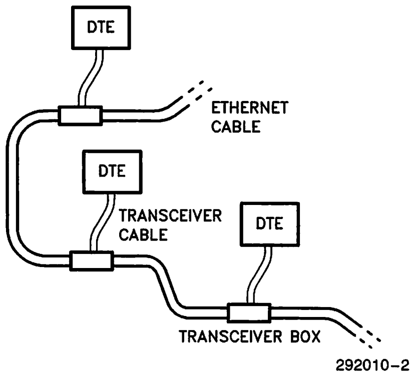


292010-5

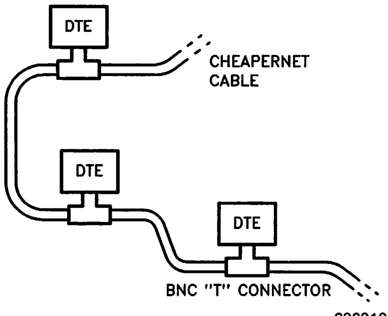


292010-6

Figure 2. 82586/82501/82502 in Ethernet and Cheapernet



292010-2



292010-3

Figure 3. Ethernet Cabling vs Cheapernet Cabling

Table 1. Differences between Ethernet and Cheapernet

	Ethernet (10BASE5)	Cheapernet (10BASE2)
Data Rate	10 M bits/sec.	10 M bits/sec.
Baseband or Broadband	Baseband (Manchester)	Baseband (Manchester)
Cable Length per Segment	500m	185m
Nodes per Segment	100	30
Node Spacing	2.5m	0.5m
Cable Type	0.4 in diameter 50Ω Double Shielded example: Ethernet Coax.	0.2 in diameter 50Ω Single or Double Shielded example: RG 58 A/U or RG 58 C/U
Transceiver Cable	Yes, up to 50m	No, not needed
Capacitance per node	4 pF	8 pF
Typical Connector	Clamp-on Tap Connector or Type N Plug Connector	BNC Female Connector

Because of the lower quality cables and connectors used in Cheapernet, there are some drawbacks. The maximum distance for one Cheapernet cable segment is only 185m (600 feet), whereas 500m (1640 feet) for Ethernet. The maximum number of nodes allowed for one Cheapernet cable segment is 30. Ethernet on the other hand allows the maximum of 100 nodes per segment. A BNC "T" connector used in Cheapernet introduces more electrical discontinuity on the transmission line than the clamp-on tap connector widely used for Ethernet. The maximum capacitance load allowed at a Cheapernet connection is 8 pF, while it is 4 pF for Ethernet. The differences are summerized in Table 1.0.

Since Ethernet and Cheapernet share the same functional and electrical characteristics, both may be mixed in a network as shown in Figure 4. In this hybrid Ethernet/Cheapernet network, it is important to keep the network propagation delay within 46.4 μs. The network may be expanded as required within this round trip propagation delay limit. Ethernet, for example, may serve as a backbone for Cheapernet in a hybrid Ethernet/Cheapernet network.

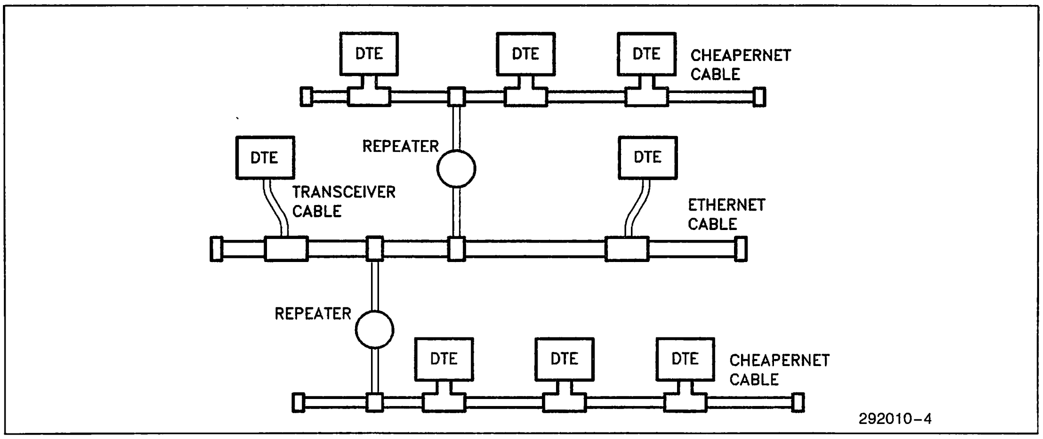


Figure 4. Ethernet/Cheapernet Hybrid Network

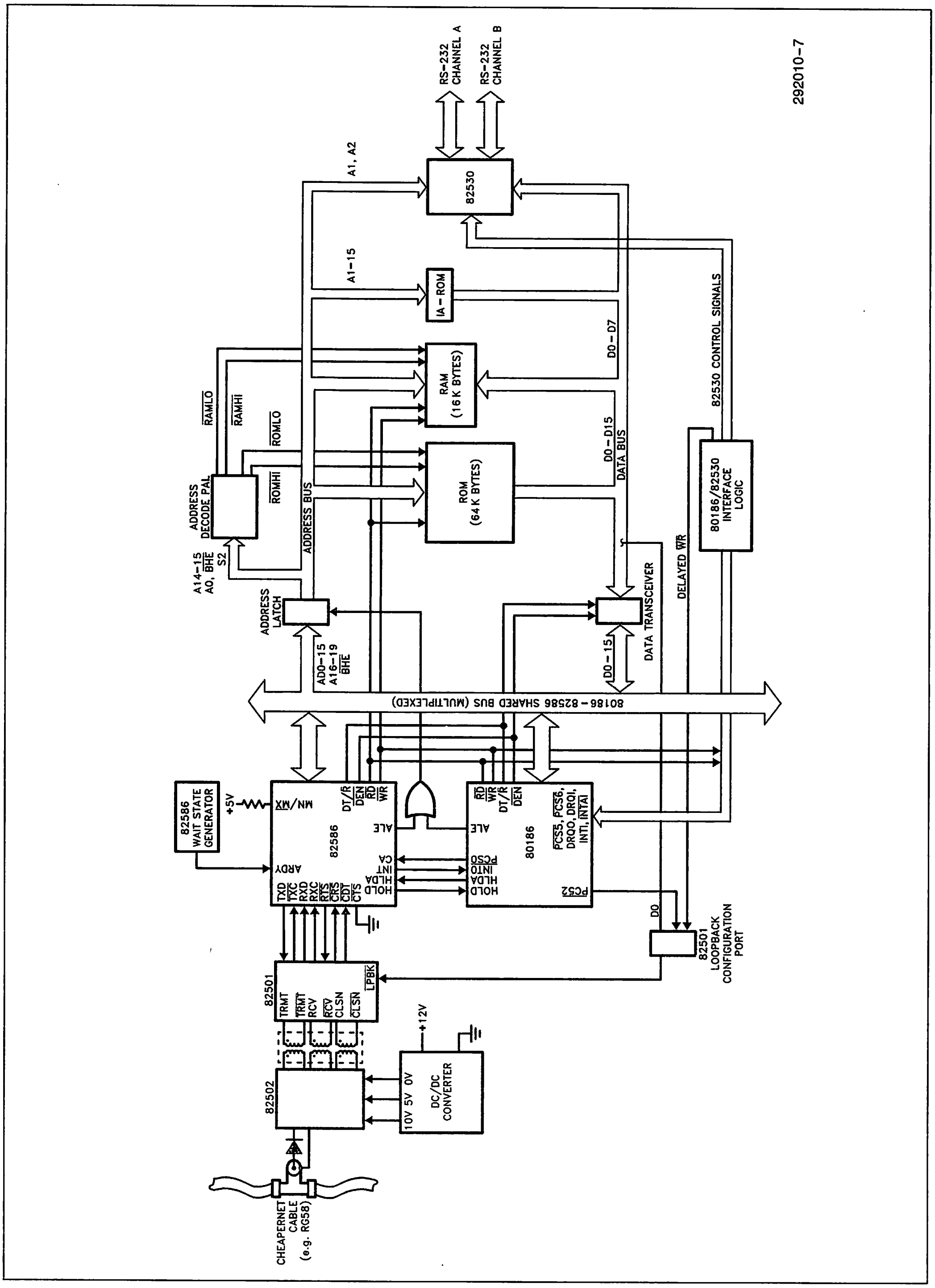


Figure 5. LANHIB Block Diagram

3.0 ETHERNET/CHEAPERNET NODE DESIGN

Details on LAN High Integration Board (LANHIB) design are presented in this section. The LANHIB is an 82586/80186 shared bus board and can be configured to Ethernet or Cheapernet. The 82586 is used in minimum mode to reduce chip count.

The reader is advised to refer to the 80186, 82586, 82501, and 82502 data sheets. Basic understanding of the 80186 microprocessor is assumed. Figure 5 shows the block diagram of the LANHIB. Schematics are in Appendix A.

3.1 82586 (Min Mode) Interface to the 80186

The 82586 can be placed in minimum mode by strapping the MN/$\overline{\text{MX}}$ pin to V_{CC}. In the minimum mode, the chip directly provides all bus control signals—ALE, $\overline{\text{RD}}$, $\overline{\text{WR}}$, DT/$\overline{\text{R}}$, and $\overline{\text{DEN}}$, saving the 8288 Bus Controller. The 80186, which is the only other bus master on the shared bus, also generates these bus control signals directly. The HOLDs and HLDAs of these two chips are connected together so that only one of the two bus masters can exclusively drive the bus at a time under the HOLD/HLDA protocol. Except for the ALE, all bus signals including address and data lines float when the chip does not have control of the bus. In this design example, $\overline{\text{RD}}$s, $\overline{\text{WR}}$s, DT/$\overline{\text{R}}$ and $\overline{\text{DEN}}$ from the two chips are connected together respectively. ALEs from the two chips are connected to an OR-gate to generate a system ALE. Multiplexed address data lines AD0–AD15 and address lines A15–A19 of the two chips are also connected line by line correspondingly.

3.2 82586 Address Latch Interface

Figure 6 shows the timing of the address signals with respect to the ALE signal. The ALE of the 82586 is OR-ed with the ALE of the 80186 and the result is connected to the latch enable inputs of Octal Transceiver Latches. The latches transfer the input data to the output as long as the latch enable is high, and captures the input data into the latch when the latch enable goes low. In this timing diagram, the setup and hold times of the input data (82586 address) required by the address latch can be verified. Estimating 7 ns of propagation delay in the 74S32, the setup time is T38 + 7, which is 32 ns at 8 MHz. The hold time for A19 is shorter than the other address lines because it is valid only during T1. The hold time for the A19 is T4 − T36 − 7, which is 3 ns. The hold time for the other address lines is T39 − 7, which is 38 ns. In this design, a 74F373 was chosen to latch address lines A16–A19 and two 74LS373s were used to latch address lines AD0–AD15. Required setup and hold times of the 74F and 74LS 373s are summerized in Table 2.

Note that address lines A16–A18 and $\overline{\text{BHE}}$ of the 82586 are not really needed to be latched. These lines stay valid for an entire memory cycle.

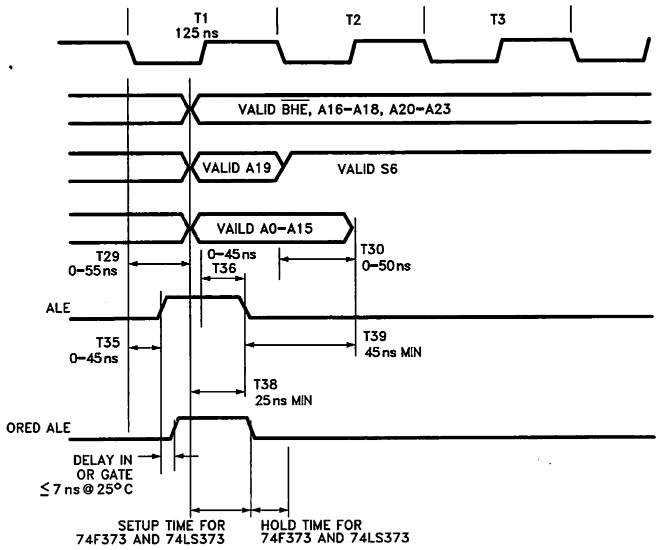

T1
125 ns
T2
T3
VALID $\overline{\text{BHE}}$, A16–A18, A20–A23
VALID A19
VALID S6
VAILD A0–A15
T29
0–55 ns
0–45 ns
T36
T30
0–50 ns
ALE
T35
0–45 ns
T39
45 ns MIN
T38
25 ns MIN
ORED ALE
DELAY IN OR GATE ≤7 ns @ 25°C
SETUP TIME FOR 74F373 AND 74LS373
HOLD TIME FOR 74F373 AND 74LS373
292010-8

Figure 6. 82586 Address Timing

Table 2. 74F and 74LS Data Setup and Hold Time Specifications at 25°C

	74F373			74LS373			Unit
	Min	Nom	Max	Min	Nom	Max	
Data Setup Time	2↓			5↓			ns
Data Hold Time	3↓			20↓			ns

3.3 80186 Address Latch Interface

The address latch used by the 82586 is shared by the 80186. Figure 7 shows the 80186 address line timing with respect to the ALE. Again estimating 7 ns delay in the 74S32, the setup time for the latch is TAVAL + 7 and the hold time is TLLAX − 7. These are 37 ns and 23 ns respectively at 8 MHz. Comparing to the required values shown in Table 2, it is quite obvious that the setup and hold times of the latch are met by wide margins. Note that the 80186's address lines A16–A18 and $\overline{\text{BHE}}$ are not valid for an entire memory cycle; therefore, they have to be latched.

3.4 82586 Memory Interface

The 74LS373 has a delay of 18 ns for input data to reach the output assuming the latch enable is high. A demultiplexed valid address (output of the address latch), therefore, becomes available after T29 + 18 measuring from the beginning of T1 (Figure 8). The demultiplexed address remains valid until the ALE of the next memory access becomes active. Upper address lines, A14 through A20, are connected to a 16L8 PAL, which provides address decode logic for all memory devices. The PAL truth table is in Appendix A. The PAL has a maximum of 35 ns propagation delay, so chip selects will become active after 55 + 18 + 35 ns (max.) from the beginning of T1 as indicated in Figure 8. Since address decode logic is implemented by a PAL, any memory expansion would only require a reprogramming of this PAL.

Two 74LS245 bus transceiver chips are controlled by the DT/$\overline{\text{R}}$ and $\overline{\text{DEN}}$. Output enable and disable times of the 74LS245 are 40 and 25 ns respectively. The maximum propagation delay when the output enable is active is 12 ns.

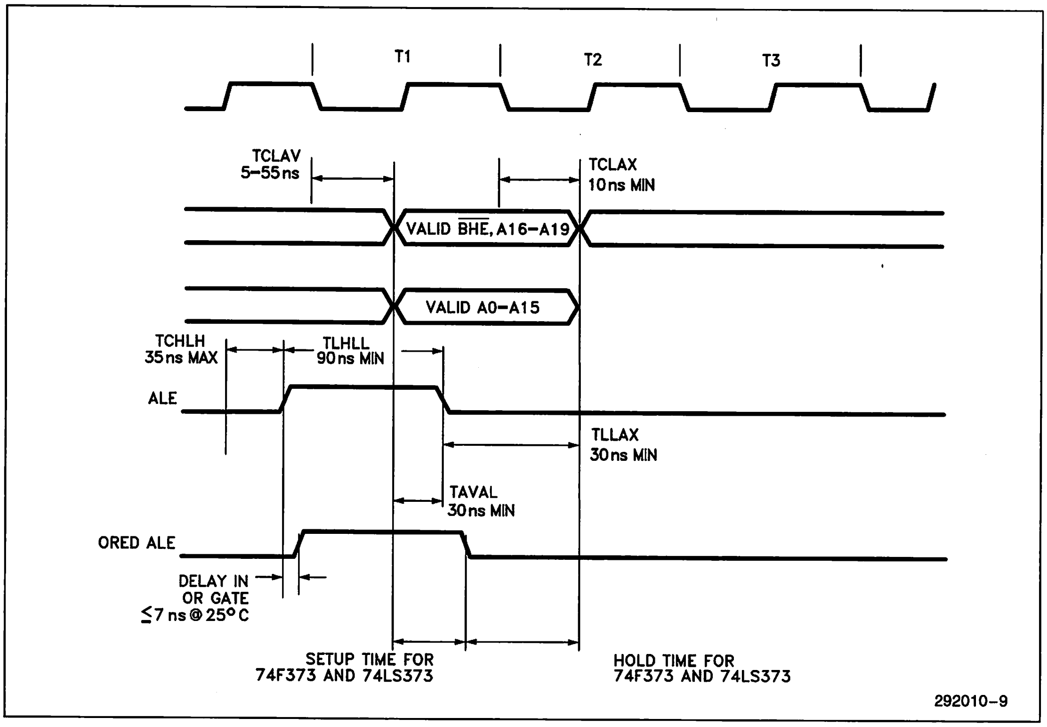


Figure 7. 80186 Address Timing

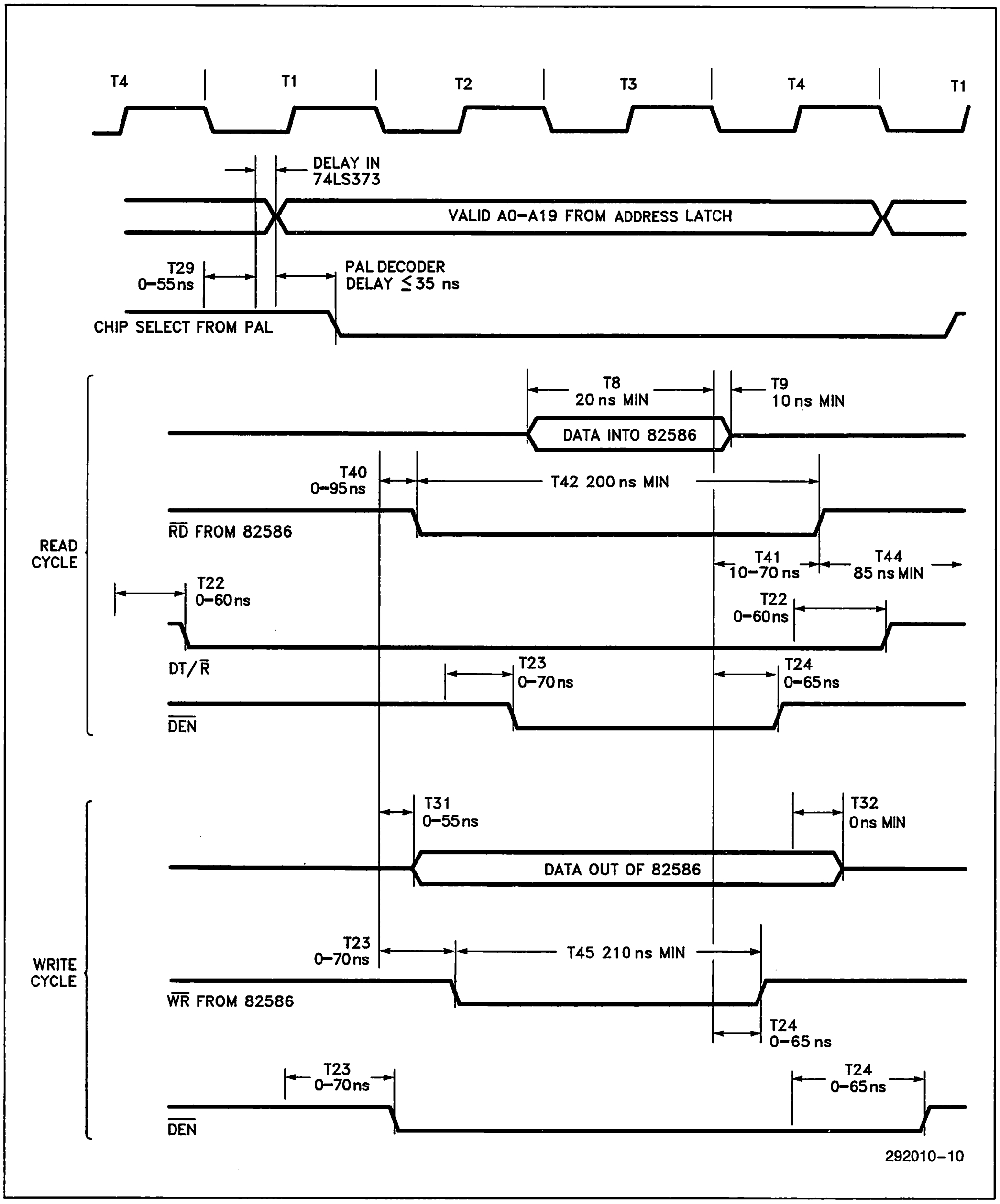


Figure 8. 82586 Memory Interface Timing

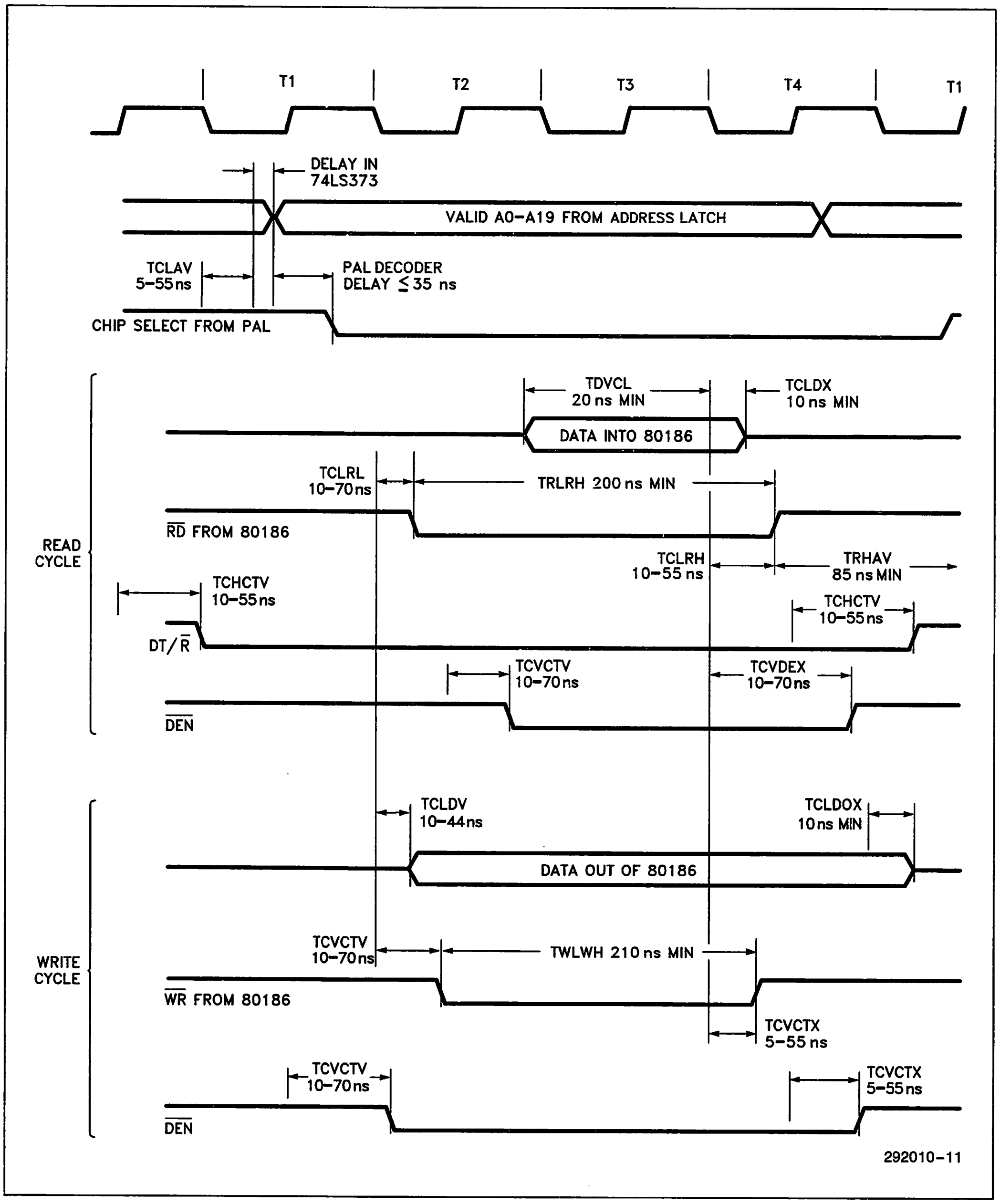


Figure 9. 80186 Memory Interface Timing

Address access time is 3 × T1 − T29 − 18 − T8 − 12 + n × T1, where n is the number of wait states. For 0 wait states operation at 8 MHz, it is 270 ns minimum. Chip select access time is 3 × T1 − T29 − 18 − T8 − 12 + n × T1 − 35, which is 235 ns for 0 wait state operation. Command access time for a read cycle is 2 × T1 − T40 − T8 − 12 + n × T1, which is 123 ns. Address setup time for a write cycle is T1 − T29 − 18 + T23, which is 52 ns minimum.

To meet these timing requirements, 2764-20s must be used for ROM. Static RAM chips, HM6264P-15, offer very wide timing margins and were selected for this design.

3.5 80186 Memory Interface

Figure 9 shows the timing of the 80186 memory interface. By comparing this figure to Figure 7, it is easy to notice that the 80186 offers a little faster bus interface. For example, TCLRL which is equivalent to T40 (0 to 95 ns) of the 82586 is specified as 10 to 70 ns. Since the memory choice satisfies the 82586 memory timing parameters, it also satisfies the 80186 memory timing parameters.

3.6 Memory Map

With 2764-20 EPROMs and 6264P-15 SRAMs, this board has 32 K bytes of ROM space and 16 K bytes of RAM space. Memory map is given in Figure 10. If 27128-20 EPROMs are used, the ROM space becomes 64 K bytes.

3.7 80186 I/O Interface

3.7.1 82586 CHANNEL ATTENTION GENERATION

The active low Peripheral Chip Select 0 ($\overline{\text{PCS0}}$) was used to generate a channel attention (CA) signal to the 82586. This way of CA generation satisfies the requirement that the width of a CA which must be wider than a clock period of the system clock.

3.7.2 82586 HARDWARE RESET PORT

$\overline{\text{PCS1}}$ of the 80186 will reset the 82586 if any I/O command is executed using this I/O chip select.

3.7.3 82530 INTERFACE

82530 interface to the 80186 was derived from the design example presented in the 82530 SCC-80186 Interface Ap Brief. This document is attached to this Ap Note as Appendix C.

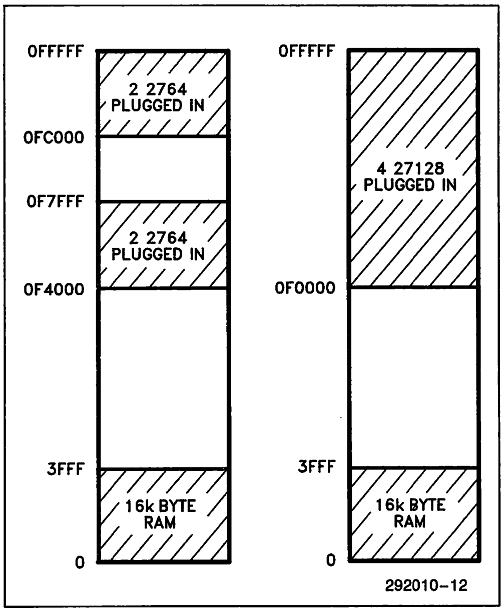


Figure 10. LANHIB Memory Map

3.7.4 82501 LOOPBACK CONFIGURATION PORT

A 74LS74 D-type flip flop was used for this port. On power up, it configures the 82501 to Non-Loopback mode by providing a high level to pin 3 ($\overline{\text{LOOPBACK}}$). The chip select is generated from the 80186's $\overline{\text{PCS2}}$ and the sychronized $\overline{\text{WR}}$ command of the 82530 interface. The least significant bit of I/O output data becomes the state of the 82501's pin 3.

3.7.5 ON-BOARD INDIVIDUAL ADDRESS PORT

To provide the 82586 a hardware configured host address, a 32x8 ROM is connected to the bus. The chip select for this ROM is generated from the 80186's $\overline{\text{PCS3}}$, so that the address for the ROM is mapped into the I/O space. Six or two (IEEE 802.3 specified address lengths) consecutive I/O reads starting from the lowest address of ROM will transfer the board address stored in the ROM to an IA-Setup command block of the 82586.

3.8 82586 Ready Signal Generation

82586 asynchronous ready (ARDY) signal is generated from a shift register. The shift register provides the 82586 a "normally ready" signal. When a wait state is needed, the ready signal is dropped to the low state. As shown in Table 3, the 82586 can be programmed to have 0 to 8 wait states by setting the DIP switch properly. Even though the on-board memory devices are

Table 3. DIP Switch Settings for Various Numbers of 82586 Wait States

Dip Switch Setting								Number of Wait States
7	6	5	4	3	2	1	0	the 82586 Inserts
1	1	1	1	1	1	1	1	0
1	1	1	1	1	1	1	0	1
1	1	1	1	1	1	0	0	2
1	1	1	1	1	0	0	0	3
1	1	1	1	0	0	0	0	4
1	1	1	0	0	0	0	0	5
1	1	0	0	0	0	0	0	6
1	0	0	0	0	0	0	0	7
0	0	0	0	0	0	0	0	8

1 = Switch Open
0 = Switch Closed

fast enough for 0 wait states operation, this programmable wait state capability was added so that the effect of wait states on the 82586 performance could be evaluated.

3.9 82501 Circuits

Since the 82501 is designed to work with the 82586, no interfacing circuits are required.

The transceiver cable side of the 82501 requires some passive components. The receive and collision differential inputs must be terminated by 78Ω ±5% resistors. Common mode voltages on these differential inputs are established internally. 240Ω ±5% pull down resistors must be connected on the TRMT and $\overline{\text{TRMT}}$ output pins.

A 0.022 μF ±10% capacitor connected between pin 1 and 2 of the 82501 is for the analog phase-locked loop.

Connected between the X1 and X2 pins is a 20 MHz parallel resonant quartz crystal (antiresonant with 20 pF load fundamental mode). An internal divide-by-two counter generates the 10 MHz clock. Since both Ethernet and Cheapernet tolerate an error of only ±0.01% in bit rate, a high quality crystal is recommended. The accuracy of a crystal should be equal to or better than ±0.002% @ 25°C and ±0.005% for 0–70°C. A 30–35 pF capacitor is connected from each crystal pin (X1 and X2) to ground in order to adjust effective capacitance load for the crystal, which should be about 20 pF including stray capacitance.

3.10 82502 Circuits

3.10.1 ISOLATION AND POWER REQUIREMENTS

The IEEE 802.3 standard requires an electrical isolation within the transceiver (MAU). Cheapernet (10BASE2) requires the isolation means to withstand 500V ac, rms for one minute. Ethernet (10BASE5) requires 250 Vrms. This electrical isolation is normally accomplished by transformer coupling of each signal pair. The kind of transformers recommended for the 82502 are the pulse transformers which have a 1:1 turn ratio and at least 50 microhenry inductance. PE64102 and PE64107 manufactured by Pulse Engineering are found to be good selections for this purpose. The PE 64102 offers 500 Vrms isolation. The PE64107 offers 2000 Vrms isolation. Both products provide three transformers in one package. Even though the current Type 10BASE5 specification requires only 250 Vrms, it is very common to have a higher isolation, at least 500 Vrms, in transceivers.

The standard specifies the voltage input level and maximum current allowed on the power pair of the transceiver cable. The voltage level may be between +11.28V dc and +15.75V dc. The maximum current is limited to 500 mA. Since the 82502 requires +10V ±10% and +5V ±10% as power, there has to be a DC/DC converter. In addition the DC/DC converter must be isolated due to the requirement described above. The DC/DC converter should be able to supply about 100 mA on the +10V line and 60 mA on the 5V line. The efficiency required in the converter is, therefore, ((11V × 100 mA + 5.5V × 60 mA) / ((11.28V − 0.5A × 4Ω) × 500 mA)) × 100 = 31% worst case. 4Ω is the maximum round trip resistance the power pair may have. 82502's CMOS process is the major contributor to this low DC/DC efficiency requirement.

Since the DC/DC converter has an isolation transformer inside, the output voltages are all floating voltages. The 0V output of the converter, for example, has no voltage relationship with the DTE's ground. The V_{SS} and AV_{SS} pins of the 82502 should be connected to the 0V output of the DC/DC converter which is the 82502's ground (reference voltage).

Both Pulse Engineering and Reliability Incorporated produce DC/DC converters that meet the 82502's requirements. The Pulse Engineering's part number is PE64369 (enclosed in this design kit). The device measures about 1.5" x 1.5" x 0.5" and provides 2000 Vrms breakdown. The Reliability's part number is 2E12R10-5. Preliminary data sheets are available from Reliability.

3.10.2 OTHER PASSIVE AND ACTIVE DEVICES FOR THE 82502

A 78Ω ±5% resistor is required to terminate the transmit pair of the Transceiver cable. The chip has an internal circuit that establishes a common mode voltage, thus no voltage divider is required. The receive and collision pair drivers need pull up resistors. A 43.2 ±1% resistor must be connected from each output pin to +5V.

A 243Ω ±0.5% precision resistor is required on the REXT pin to the ground. The accuracy of this resistor is very important since this resistor is a part of current and voltage reference circuits in the analog sections of the 82502.

Grounding the HBD (Heartbeat Disable) pin will allow the chip to perform Signal Quality Error check (Heartbeat) as required by the IEEE 802.3. The chip will transmit the collision presence signal after each transmission during Interframe Spacing (IFS) time. In a repeater application, this feature is disabled (HBD = +5V).

Diodes connected on the CXTD pin are to reduce the capacitive loading onto the coaxial cable. One diode is sufficient, but two will provide a protection in case one burns out (Short Circuit). The diode should have about 2 pF shunt capacitance at Vd = 0V and be able to handle at least 100 mA when biased in forward direction. A few candidates are 1N5282, 1N3600, and 1N4150.

A 100Ω fusible resistor connected on the CXRD pin is purely for protection. It is there as a fuse, not as a resistor. The 82502 works without this resistor. The IEEE 802.3, however, states that "component failures within the MAU (Media Attachment Unit or Transceiver) electronics should not prevent communication among other MAUs on the coaxial cable." It is recommending a transceiver design that minimizes the probability of total network failure. The fusible resistor will provide an open circuit in an event of excess current. A short circuit from the CXRD pin to ground will not bring down the network due to the blown fuse.

An 8 MΩ resistor connected between the coaxial cable shield and the Transceiver cable shield will provide a static discharge path. The Ethernet coaxial cable should also have an effective earth ground at one point in a network as required by the standard. A 0.01 μF in parallel to the 8 MΩ resistor provides ground for RF signals.

3.10.3 LAYOUT CONSIDERATION FOR THE 82502 CIRCUITS

It is strongly recommended that the board have a special ground plane for the 82502 (see Figure 11). The 0V (reference) output of the isolated DC/DC converter should be connected to the ground plane. The V_{SS} and AV_{SS} pins of the 82502 should be connected to the ground plane with minimum lead wires.

There should be a 0.22 μF capacitor connected between the coaxial cable shield and ground. The signal path from the coax. shield through the 0.22 μF capacitor to

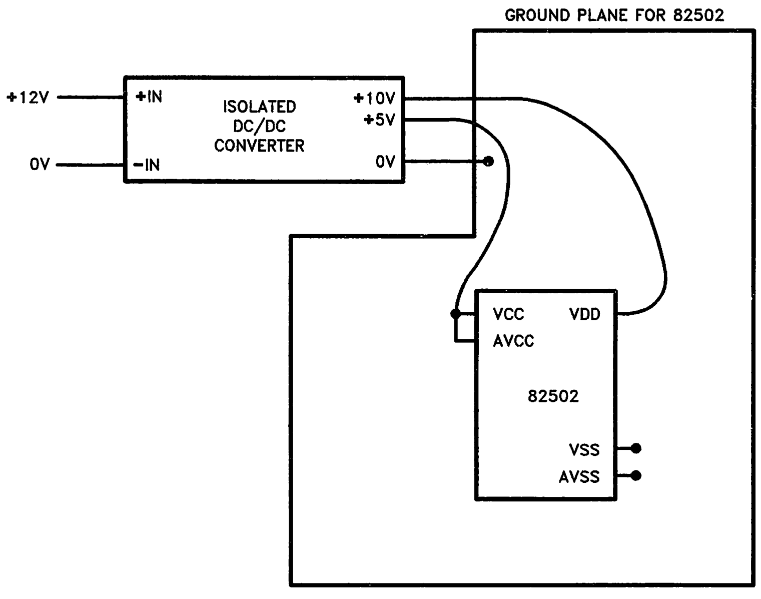


Figure 11. Ground Plane for the 82502

the ground should be kept as short as possible—leads of the 0.22 μF capacitor should be as short as possible.

The path length from the CXTD pin through two diodes to the center conductor of the coax should also be minimized.

These are recommendations which will produce a more reliable circuit if followed carefully. Remember that the 82502 has analog circuits in it.

4.0 DEMONSTRATION SOFTWARE

The demonstration software included in this Ap Note is called "Traffic Simulator and Monitor Station" (TSMS) program. The TSMS program is written in PL/M and has the following features:

1. Programmable network load generation
2. Network statistical monitoring capabilities
3. Interactive command execution of all 82586 commands
4. Interactive buffer monitoring

The environment created with the TSMS program was found to be very useful for network debugging and other individual station's hardware and software debugging. The TSMS software listing is found in Appendix B.

The 82586 Data Link Driver presented in Application Note 235 (Chapter 4 of the LAN Components User's Manual) currently runs only on the iSBC 186/51. The software will be modified to run on the LANHIB and made available as another demonstration software.

4.1 Programming PROMs to Run the TSMS Program

By returning the card enclosed in this kit or by contacting Insite, the TSMS program and related submit files can be obtained on a diskette. Files that are on the diskette are:

```
TSMS.PLM
IO.PLM
INI186.PLM
LANHIB.CSD
SBC.CSD
IUPHIB.CSD
IUPSBC.CSD
HI.BYT
LO.BYT
ROM.CSD
```

HI.BYT and LO.BYT are the files which can be downloaded to PROMs directly. These files are already configured for the LANHIB. The submit file ROM.CSD invokes the Intel PROM Programming Software (iPPS) under the ISIS-II operation system and programs two 2764 EPROMs. The Intel Universal Programmer must be placed in ON-LINE mode.

Other files contained in the diskette are for compiling and locating the original TSMS program. Using these files, the original TSMS program can be changed or can be compiled for an iSBC 186/51. 'TSMS.PLM' is the original TSMS source program. 'IO.PLM' contains the IO driver needed when the TSMS program is run on the iSBC 186/51. INI186.PLM is the LANHIB initialization routine. LANHIB.CSD is the submit file that compiles, links, and locates the TSMS program and the LANHIB initialization routine. SBC.CSD compiles, links, and locates the TSMS program and the IO driver for the iSBC 186/51. IUPHIB.CSD programs two 2764s for the LANHIB. IUPSBC.CSD programs two 2764s for the iSBC 186/51.

Therefore, if the TSMS program is to be run on the LANHIB (Demo board), steps required are:

1. submit LANHIB
2. submit IUPHIB

If the TSMS program is to be run on the iSBC 186/51, steps required are:

1. submit SBC
2. submit IUPSBC

4.2 Capabilities and Limits of the TSMS Program

The TSMS program initializes the LANHIB Ethernet/Cheapernet station by executing 82586's Diagnose, Configure, IA-Setup, and MC-Setup commands. The program asks a series of questions in order to set up a linked list of these 82586 commands. After initialization is completed, the program automatically starts the 82586's Receive Unit (monitoring capability). Transmissions are optional (traffic simulation capability).

The TSMS program has two modes of operation: Continuous mode and Interactive Command Execution mode. The program automatically gets into the Continuous mode after initialization. The Interactive Command Execution mode can be entered from the Continuous mode. Once entered in the Continuous mode, the software uses the format shown in Figure 12 to display information. Detailed description of each of these fields is as follows:

Host Address: host (station) address used in the most recently prepared IA-Setup command. The software simply writes the address stored in the IA-Setup command block with its least significant bit being in the most right position. Note that if the IA-Setup com-

```
****************************** Station Configuration *************************

  Host Address: 00  AA  00  00  18  6D
  Multicast Address(es): No Multicast Addresses Defined
  Destination Address: FF  FF  FF  FF  FF  FF
  Frame Length: 118 bytes
  Time Interval between Transmit Frames:  159.4 microseconds
  Network Percent Load generated by this station: 35.7 %
  Transmit Frame Terminal Count: Not Defined
  82586 Configuration Block: 08  00  26  00  60  00  F2  00  00  40

******************************* Station Activities ***************************

  # of Good     # of Good    CRC           Alignment    No            Receive
  Frames        Frames       Errors        Errors       Resource      Overrun
  Transmitted   Received                                Errors        Errors
  10130         0            0             0            0             0
                                                                   292010-14
```

Figure 12. Continuous Mode Display

mand was just set up and not executed, the address displayed in this field may not be the address stored in the 82586.

Multicast Address(es): multicast addresses used in the most recently prepared MC-Setup command. As in the case of host address, the software simply writes the addresses stored in the MC-Setup command block. Note that if the MC-Setup command was just set up and not executed, the addresses displayed in this field may not be the addresses stored in the 82586.

Destination Address: destination address stored in the transmit command block if AL-LOC=0. If AL-LOC=1, destination address is picked up from the transmit buffer. The least significant bit is in the most right position.

Frame Length: transmit frame byte count including destination address, source address, length, data, and CRC field.

Time Interval Between Transmit Frames: approximate time interval obtainable between transmit frames (Figure 13). The number is correct if there are no other stations transmitting on the network.

Network Percent Load Generated by This Station: approximate network percent load that is generated by this station (Figure 13). The number is correct if there are no other stations transmitting on the network.

Transmit Frame Terminal Count: number of frames this station will transmit before it stops network traffic load generation. If this station is transmitting indefinitely, this field will be 'Not Defined'.

82586 Configuration Block: configuration parameters used in the most recently prepared Configure command. As in the case of IA-Setup command, the software simply writes the parameters from the Configure command block. The least significant byte (FIFO Limit) of the configuration parameters is printed in the most left position.

# of Good Frames Transmitted: number of good frames transmitted. This is a snap shot of the 32-bit transmit frame counter. It is incremented only when both C and OK bits of the transmit command status are set after an execution. The counter is 32-bit wide.

# of Good Frames Received: number of good frames received. This is a snap shot of the 32-bit receive frame counter. It is incremented only when both C and OK bits of a receive frame descriptor status are set after a reception. The counter is 32-bit wide.

CRC Errors: number of frames that had a CRC error. This is a snap shot of the 16-bit CRC counter maintained by the 82586 in the SCB.

Alignment Errors: number of frames that had an alignment error. This is a snap shot of the 16-bit alignment counter maintained by the 82586 in the SCB.

No Resource Errors: number of frames that had a no resource error. This is a snap shot of the 16-bit no resource counter maintained by the 82586 in the SCB.

Receive Overrun Errors: number of frames that had a receive overrun error. This is a snap shot of the 16-bit receive overrun error counter maintained by the 82586 in the SCB.

If the station is actively transmitting, # of good frames transmitted should be incrementing. If the station is actively receiving frames, # of good frames received should be incrementing. In this continuous mode, a user can see the activities of the network.

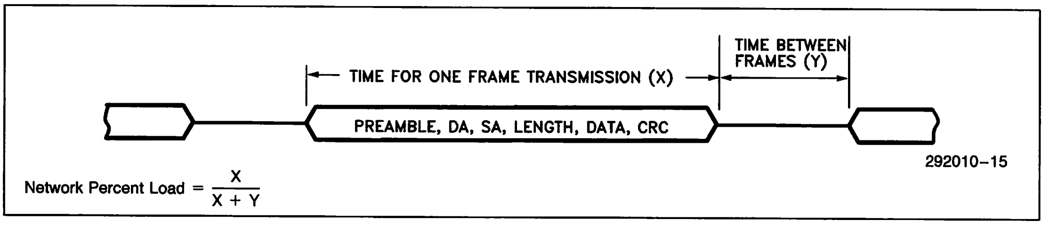

TIME BETWEEN FRAMES (Y)
TIME FOR ONE FRAME TRANSMISSION (X)
PREAMBLE, DA, SA, LENGTH, DATA, CRC
292010-15
Network Percent Load = $\frac{X}{X + Y}$

Figure 13. Network Percent Load

Hitting any key on the keyboard while the program is running in the Continuous mode will exit the mode. The program will respond with a message 'Enter Command (H for Help) → '. In this Interactive Command Execution mode, a user can set up any one of the 82586 action commands and/or execute any one of the 82586 SCB control commands. Setting up a Dump command and executing a SCB Command Unit Start command will, for example, execute the Dump command. Display commands are also available to see the contents of the 82586's data structure blocks. A display command will enable a user to see the contents of the 82586's dump (see Section 6.3).

Typing 'E' after 'Enter command (H for help) → ', executing a SCB Command Unit Start command with a transmit command, or executing a SCB Receive Unit Start command will exit the Interactive Command Execution mode. The program will be back in the Continuous mode. Using this Interactive Command Execution mode, one can, for example, reconfigure the station and come back to the Continuous mode. Section 6 lists actual example executions of the TSMS program.

The TSMS program should be run in an 8 MHz system. The software running at 8 MHz with a maximum of 2 wait states has been tested and verified to be able to receive back-to-back frames separated by 9.6 microseconds and still keep track of the correct number of frames received. This capability, for example, can be used to find out exactly how many frames a new station in the network had transmitted.

The software does not perform extensive loopback tests and hardware diagnostics during the initialization. A loopback operation can be performed interactively in the Interactive Command Execution mode.

The software allows a user to set up only 8 multicast addresses maximum. It is not possible with this program to set up more than 8 multicast addresses.

The command chaining feature of the 82586 is not used in the Interactive Command Execution mode. Each command setup performed by a 'S' command after 'Enter command (H for help) → ' sets up a command with its EL bit set, I bit reset, and S bit reset. Diagnose, Configure, IA-Setup, and MC-Setup commands are chained together during the initialization routine and executed at once with only one CA.

The software sets up 5 Receive Frame Descriptors linked in a circular list. Therefore, a user can see only the last 5 frames the station has received. It also sets up 5 receive buffers, each being 1514 bytes long, linked in circle. Therefore, the 82586 never goes into the NO RESOURCES state.

4.3 Example Executions of the TSMS Program

This section presents three example executions of the TSMS program. When the TSMS program needs a command to be typed, it asks a question with ' → '. Anything after ' → ' is what a user needs to type in on the keyboard. To switch from the continuous mode to the interactive command execution mode, type any key on the keyboard.

4.3.1 EXAMPLE 1: EXTERNAL LOOPBACK EXECUTION

In this example, 500 external loopback transmissions and receptions are executed (Figure 14). In order for the software to process each loopback properly, a large delay was given between transmissions.

4.3.2 EXAMPLE 2: FRAME RECEPTION IN PROMISCUOUS MODE

The 82586 is configured to receive any frame that exists in the network (Figure 15). In this example, the station received 100 frames.

4.3.3 EXAMPLE 3: 35.7% NETWORK TRAFFIC LOAD GENERATION

The station is programmed to transmit 118 byte long frames with a time interval of 159.4 microseconds in between (Figure 16). The network load is about 35.7 percent if no other stations are transmitting in the network.

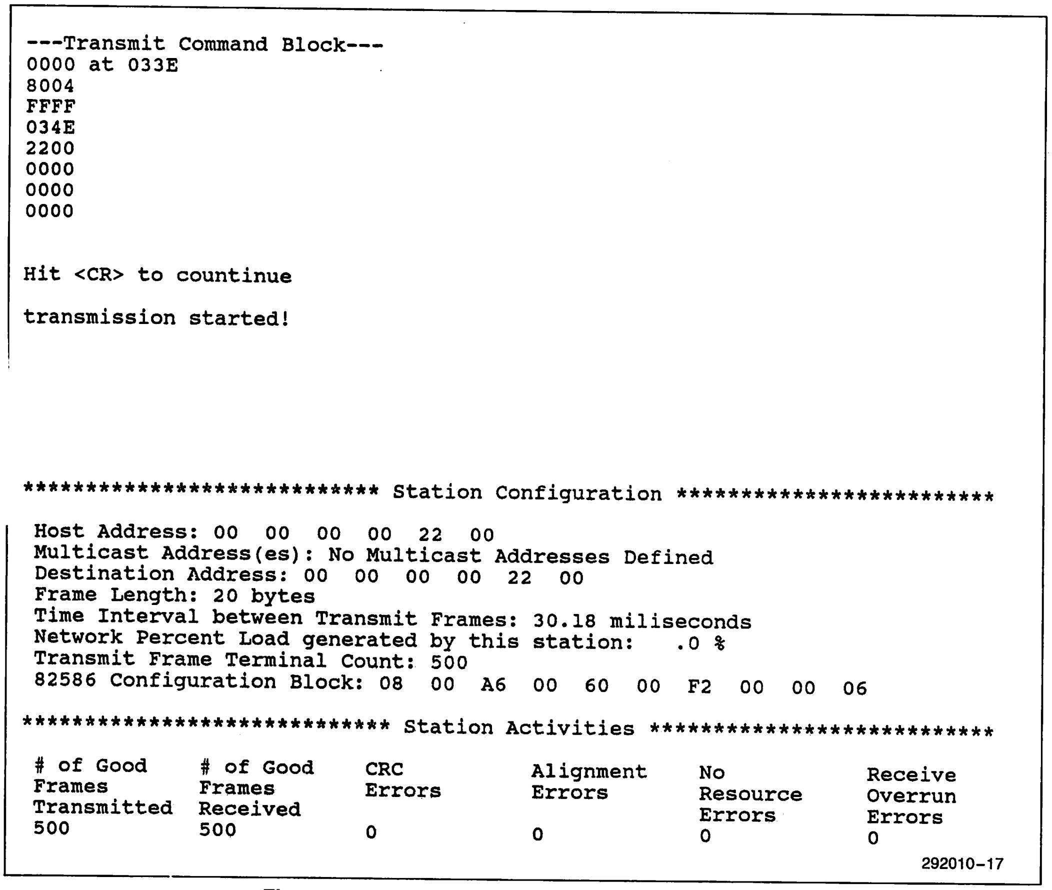
A key was hit to enter the Interactive Command Execution mode. In that mode, a Dump command was executed and the result was displayed. After the Dump execution, a transmit command was set up again and the station was put in the Continuous mode.

Traffic Simulator and Monitor Station Program

```
Initialization begun


Configure command is set up for default values.
Do you want to change any bytes? (Y or N) ==> Y
Enter byte number (1 - 11) ==> 4
Enter byte 4 (4H) ==> A6H
Any more bytes? (Y or N) ==> Y
Enter byte number (1 - 11) ==> 11
Enter byte 11 (BH) ==> 6
Any more bytes? (Y or N) ==> N
Configure the 586 with the prewired board address ==> N
Enter this station's address in Hex ==> 000000002200
You can enter up to 8 Multicast Addresses.
Would you like to enter a Multicast Address? (Y or N) ==> N
You entered 0 Multicast Address(es).

Would you like to transmit?
Enter a Y or N ==> Y
Enter a destination address in Hex ==> 000000002200

Enter TYPE ==> 0
How many bytes of transmit data?
Enter a number ==> 2
Transmit Data is continuous numbers (0, 1, 2, 3, ... )
Change any data bytes? (Y or N) ==> N
Enter a delay count ==> 10000000000
The number is too big.
It has to be less than or equal to 65535 (FFFFH).
Enter a number ==> 60000

Setup a transmit terminal count? (Y or N) ==> Y
Enter a transmit terminal count ==> 500

Destination Address: 00  00  00  00  22  00
Frame Length: 20 bytes
Time Interval between Transmit Frames: 30.18 miliseconds
Network Percent Load generated by this station:   .0 %
Transmit Frame Terminal Count: 500

Good enough? (Y or N) ==> Y

Receive Unit is active.
```

292010-16

Figure 14. External Loopback Execution



```
---Transmit Command Block---
0000 at 033E
8004
FFFF
034E
2200
0000
0000
0000


Hit <CR> to countinue

transmission started!




****************************** Station Configuration *************************

 Host Address: 00  00  00  00  22  00
 Multicast Address(es): No Multicast Addresses Defined
 Destination Address: 00  00  00  00  22  00
 Frame Length: 20 bytes
 Time Interval between Transmit Frames: 30.18 miliseconds
 Network Percent Load generated by this station:   .0 %
 Transmit Frame Terminal Count: 500
 82586 Configuration Block: 08  00  A6  00  60  00  F2  00  00  06

****************************** Station Activities ***************************

 # of Good     # of Good    CRC          Alignment    No           Receive
 Frames        Frames       Errors       Errors       Resource     Overrun
 Transmitted   Received                               Errors       Errors
 500           500          0            0            0            0
```

292010-17

Figure 14. External Loopback Execution (Continued)

Traffic Simulator and Monitor Station Program

```
 Initialization begun


 Configure command is set up for default values.
 Do you want to change any bytes? (Y or N) ==> Y
 Enter byte number (1 - 11) ==> 9
 Enter byte 9 (9H) ==> 1
 Any more bytes? (Y or N) ==> N
 Configure the 586 with the prewired board address ==> Y
 You can enter up to 8 Multicast Addresses.
 Would you like to enter a Multicast Address? (Y or N) ==> N
 You entered 0 Multicast Address(es).

 Would you like to transmit?
 Enter a Y or N ==> N

 Receive Unit is active.






***************************** Station Configuration ************************

 Host Address: 00  AA  00  00  18  6D
 Multicast Address(es): No Multicast Addresses Defined
 82586 Configuration Block: 08  00  26  00  60  00  F2  01  00  40

****************************** Station Activities **************************

 # of Good     # of Good     CRC           Alignment     No            Receive
 Frames        Frames        Errors        Errors        Resource      Overrun
 Transmitted   Received                                  Errors        Errors
 0             100           0             0             0             0
 Enter command (H for help) ==> D

 Command Block or Receive Area? (R or C) ==> R
 Frame Descriptors:
4000 at 036C  A000 at 0382  A000 at 0398  A000 at 03AE  A000 at 03C4
0000          0000          0000          0000          0000
0382          0398          03AE          03C4          036C
03DA          03E4          03EE          03F8          0402
2200          2200          2200          2200          2200
2200          2200          2200          2200          2200
0000          0000          0000          0000          0000
```

292010-18

Figure 15. Frame Reception in Promiscuous Mode

```
0000          0000          0000          0000          0000
0000          0000          0000          0000          0000
0000          0000          0000          0000          0000
0000          0000          0000          0000          0000

 Receive Buffer Descriptors:
C064 at 03DA  C064 at 03E4  C064 at 03EE  C064 at 03F8  C064 at 0402
03E4          03EE          03F8          0402          03DA
040C          09F6          0FE0          15CA          1BB4
0000          0000          0000          0000          0000
05DC          05DC          05DC          05DC          05DC

 Display the receive buffers? (Y or N) ==> Y
 Receive Buffers:

 Receive Buffer 0 :
002C:014C 00  01  02  03  04  05  06  07  08  09  0A  0B  0C  0D  0E  0F
002C:015C 10  11  12  13  14  15  16  17  18  19  1A  1B  1C  1D  1E  1F
002C:016C 20  21  22  23  24  25  26  27  28  29  2A  2B  2C  2D  2E  2F
002C:017C 30  31  32  33  34  35  36  37  38  39  3A  3B  3C  3D  3E  3F
002C:018C 40  41  42  43  44  45  46  47  48  49  4A  4B  4C  4D  4E  4F
002C:019C 50  51  52  53  54  55  56  57  58  59  5A  5B  5C  5D  5E  5F
002C:01AC 60  61  62  63

Hit <CR> to countinue

 Receive Buffer 1 :
002C:0736 00  01  02  03  04  05  06  07  08  09  0A  0B  0C  0D  0E  0F
002C:0746 10  11  12  13  14  15  16  17  18  19  1A  1B  1C  1D  1E  1F
002C:0756 20  21  22  23  24  25  26  27  28  29  2A  2B  2C  2D  2E  2F
002C:0766 30  31  32  33  34  35  36  37  38  39  3A  3B  3C  3D  3E  3F
002C:0776 40  41  42  43  44  45  46  47  48  49  4A  4B  4C  4D  4E  4F
002C:0786 50  51  52  53  54  55  56  57  58  59  5A  5B  5C  5D  5E  5F
002C:0796 60  61  62  63

Hit <CR> to countinue

 Receive Buffer 2 :
002C:0D20 00  01  02  03  04  05  06  07  08  09  0A  0B  0C  0D  0E  0F
002C:0D30 10  11  12  13  14  15  16  17  18  19  1A  1B  1C  1D  1E  1F
002C:0D40 20  21  22  23  24  25  26  27  28  29  2A  2B  2C  2D  2E  2F
002C:0D50 30  31  32  33  34  35  36  37  38  39  3A  3B  3C  3D  3E  3F
002C:0D60 40  41  42  43  44  45  46  47  48  49  4A  4B  4C  4D  4E  4F
002C:0D70 50  51  52  53  54  55  56  57  58  59  5A  5B  5C  5D  5E  5F
002C:0D80 60  61  62  63

Hit <CR> to countinue

 Receive Buffer 3 :
002C:130A 00  01  02  03  04  05  06  07  08  09  0A  0B  0C  0D  0E  0F
002C:131A 10  11  12  13  14  15  16  17  18  19  1A  1B  1C  1D  1E  1F
002C:132A 20  21  22  23  24  25  26  27  28  29  2A  2B  2C  2D  2E  2F
002C:133A 30  31  32  33  34  35  36  37  38  39  3A  3B  3C  3D  3E  3F
002C:134A 40  41  42  43  44  45  46  47  48  49  4A  4B  4C  4D  4E  4F
002C:135A 50  51  52  53  54  55  56  57  58  59  5A  5B  5C  5D  5E  5F
002C:136A 60  61  62  63

Hit <CR> to countinue
```

292010-19

Figure 15. Frame Reception in Promiscuous Mode (Continued)

```
 Receive Buffer 4 :
002C:18F4 00  01  02  03  04  05  06  07  08  09  0A  0B  0C  0D  0E  0F
002C:1904 10  11  12  13  14  15  16  17  18  19  1A  1B  1C  1D  1E  1F
002C:1914 20  21  22  23  24  25  26  27  28  29  2A  2B  2C  2D  2E  2F
002C:1924 30  31  32  33  34  35  36  37  38  39  3A  3B  3C  3D  3E  3F
002C:1934 40  41  42  43  44  45  46  47  48  49  4A  4B  4C  4D  4E  4F
002C:1944 50  51  52  53  54  55  56  57  58  59  5A  5B  5C  5D  5E  5F
002C:1954 60  61  62  63

Hit <CR> to countinue

 Enter command (H for help) ==> E




****************************** Station Cofiguration **************************

 Host Address: 00  AA  00  00  18  6D
 Multicast Address(es): No Multicast Addresses Defined
 82586 Configuration Block: 08  00  26  00  60  00  F2  01  00  40

******************************* Station Activities ***************************

 # of Good     # of Good     CRC           Alignment     No            Receive
 Frames        Frames        Errors        Errors        Resource      Overrun
 Transmitted   Received                                  Errors        Errors
 0             100           0             0             0             0
```

292010-20

Figure 15. Frame Reception in Promiscuous Mode (Continued)

Traffic Simulator and Monitor Station Program

```
 Initialization begun


 Configure command is set up for default values.
 Do you want to change any bytes? (Y or N) ==> N
 Configure the 586 with the prewired board address ==> Y
 You can enter up to 8 Multicast Addresses.
 Would you like to enter a Multicast Address? (Y or N) ==> N
 You entered 0 Multicast Address(es).

 Would you like to transmit?
 Enter a Y or N ==> Y
 Enter a destination address in Hex ==> FFFFFFFFFFFF

 Enter TYPE ==> 0
 How many bytes of transmit data?
 Enter a number ==> 100
 Transmit Data is continuous numbers (0, 1, 2, 3, ... )
 Change any data bytes? (Y or N) ==> N

 Enter a delay count ==> 0

 Setup a transmit terminal count? (Y or N) ==> N

 Destination Address: FF  FF  FF  FF  FF  FF
 Frame Length: 118 bytes
 Time Interval between Transmit Frames:  159.4 microseconds
 Network Percent Load generated by this station: 35.7 %
 Transmit Frame Terminal Count: Not Defined

 Good enough? (Y or N) ==> Y

 Receive Unit is active.

---Transmit Command Block---
0000 at 033E
8004
FFFF
034E
FFFF
FFFF
FFFF
0000


Hit <CR> to countinue
```

292010-21

Figure 16. 35.7% Network Load Generation

```
transmission started!


*************************** Station Configuration ***************************

 Host Address: 00  AA  00  00  18  6D
 Multicast Address(es): No Multicast Addresses Defined
 Destination Address: FF  FF  FF  FF  FF  FF
 Frame Length: 118 bytes
 Time Interval between Transmit Frames:  159.4 microseconds
 Network Percent Load generated by this station: 35.7 %
 Transmit Frame Terminal Count: Not Defined
 82586 Configuration Block: 08  00  26  00  60  00  F2  00  00  40

**************************** Station Activities ****************************

 # of Good     # of Good     CRC          Alignment     No            Receive
 Frames        Frames        Errors       Errors        Resource      Overrun
 Transmitted   Received                                 Errors        Errors
 10459         0             0            0             0             0
 Enter command (H for help) ==> H

 Commands are:
 S - Setup CB               D - Display RFD/CB
 P - Print SCB              C - SCB Control CMD
 L - ESI Loopback On        N - ESI Loopback Off
 A - Toggle Number Base
 Z - Clear Tx Frame Counter
 Y - Clear Rx Frame Counter
 E - Exit to Continuous Mode


 Enter command (H for help) ==> S

 Enter command block type (H for help) ==> H

Command block type:
 N - Nop                 I - IA Setup
 C - Configure           M - MA Setup
 T - Transmit            R - TDR
 D - Diagnose            S - Dump Status
 H - Print this message

 Enter command block type (H for help) ==> S

 Enter command (H for help) ==> C

 Do you want to enter any SCB commands? (Y or N) ==> Y
 Enter CUC ==> 1
 Enter RES bit ==> 0
 Enter RUC ==> 0
 Issued Channel Attention



 Enter command (H for help) ==> D
```

292010-22

Figure 16. 35.7% Network Load Generation (Continued)

```
 Command Block or Receive Area? (R or C) ==> C
---Dump Status Command Block---
A000 at 0364
8006
FFFF
27D6

 Dump Status Results
 at 27D6
00  E8  3F  26  08  60  00  FA  00  00  40  FF  6D  18  00  00
AA  00  40  20  00  00  00  00  FF  FF  FF  FF  B5  9E  EE  CF
62  63  3F  B0  00  00  00  00  00  00  00  00  FF  85  08  FC
00  00  00  00  00  00  00  00  00  00  00  00  70  03  06  00
DC  05  00  00  0C  04  DC  05  E4  03  DA  03  DA  03  78  05
82  03  6C  03  F8  03  64  80  D6  27  E8  21  FF  FF  4E  03
06  80  FF  FF  64  03  00  00  D2  02  00  00  00  00  00  00
00  00  D6  27  00  01  00  28  00  00  00  00  30  26  00  00
20  00  40  06  30  01  00  00  90  00  10  01  00  00  6C  03
00  00  6A  03  0E  00  6C  28  00  00  74  03  00  00  00  00
00  00  00  00  00  C0  00  00  00  00


 Enter command (H for help) ==> S

 Enter command block type (H for help) ==> T
 Enter a destination address in Hex ==> FFFFFFFFFFFF

 Enter TYPE ==> 0
 How many bytes of transmit data?
 Enter a number ==> 100
 Transmit Data is continuous numbers (0, 1, 2, 3, ... )
 Change any data bytes? (Y or N) ==> N

 Enter a delay count ==> 0

 Setup a transmit terminal count? (Y or N) ==> N

 Destination Address: FF  FF  FF  FF  FF  FF
 Frame Length: 118 bytes
 Time Interval between Transmit Frames:  159.4 microseconds
 Network Percent Load generated by this station: 35.7 %
 Transmit Frame Terminal Count: Not Defined

 Good enough? (Y or N) ==> Y

 Enter command (H for help) ==> C

 Do you want to enter any SCB commands? (Y or N) ==> Y
 Enter CUC ==> 1
 Enter RES bit ==> 0
 Enter RUC ==> 0
 Issued Channel Attention
```

292010-23

Figure 16. 35.7% Network Load Generation (Continued)

```
*************************** Station Configuration ************************

 Host Address: 00  AA  00  00  18  6D
 Multicast Address(es): No Multicast Addresses Defined
 Destination Address: FF  FF  FF  FF  FF  FF
 Frame Length: 118 bytes
 Time Interval between Transmit Frames:  159.4 microseconds
 Network Percent Load generated by this station: 35.7 %
 Transmit Frame Terminal Count: Not Defined
 82586 Configuration Block: 08  00  26  00  60  00  F2  00  00  40

**************************** Station Activities **************************

 # of Good     # of Good    CRC          Alignment    No           Receive
 Frames        Frames       Errors       Errors       Resource     Overrun
 Transmitted   Received                               Errors       Errors
 106020        0            0            0            0            0
                                                                  292010-24
```

Figure 16. 35.7% Network Load Generation (Continued)

5.0 IN CASE OF DIFFICULTY

This section presents methods of troubleshooting ("debugging") a LANHIB board. When a LANHIB board is powered up with the TSMS program stored in EPROMs, it should display "TRAFFIC SIMULATOR AND MONITOR STATION PROGRAM" message on a terminal screen. If the message is not displayed, the board has to be debugged. Section 5.1 describes basic 80186/82586 system troubleshooting procedures. Section 5.2 is for troubleshooting 82501 and 82502 circuits. After the 80186/82586 system is debugged, the 82501/82502 circuits have to be tested.

5.1 Troubleshooting 80186/82586 System

Shown in Figure 17 is a flow chart for troubleshooting 80186/82586 system. The procedure requires an oscilloscope. A logic analyzer is needed if problems appear to be serious. The procedures will debug the board to the point where the 82530 is initialized properly. If the 82530 can be initialized properly, ROM and RAM interfaces must be functioning. Board initialization routines (INI186.PLM) linked to the TSMS program requires ROM and RAM accesses. Since the 82586 shares most of the system with the 80186, no special debugging is required for the 82586. Wiring of all 82586 parallel signal pins should, however, be checked.

The flow chart branches to two major paths after the first decision box. One path debugs the RS-232 channel and the other debugs the 80186/82586 system. The waveform of the $\overline{\text{TRXCB}}$ output of the 82530 determines which path to be taken. If the 82530 is getting programmed properly, there should be 153.6 KHz (1/f = 6.51 μs) clock on this output pin. If there is a clock, the problem is probably in the RS-232 interface. If there is no clock, then the system has to be debugged using a logic analyzer.

5.2 Troubleshooting 82501/82502 Circuits

If the TSMS program runs on the LANHIB but the 82586 is not able to transmit or receive, there must be a problem in 82501/82502 circuits. The flow chart in Figure 19 will guide troubleshooting in these circuits. An oscilloscope is required.

The board should be configured to Cheapernet and disconnected from the network. Two terminators will be required to terminate a "T" BNC connector providing an effective load resistance of 25Ω to the 82502.

The 82586 must have the system and transmit clocks running upon reset. Since the transmit clock is generated by the 82501, the 82501 transmit clock output pin (pin 16) should be checked. The TSMS program executes 82586's Diagnose, Configure, IA-Setup, and MC-Setup commands during initialization. If the 82586 has active $\overline{\text{CRS}}$ (Carrier Sense) signal, it cannot complete execution of these commands. The 82501 should, therefore, be checked if it is generating inactive $\overline{\text{CRS}}$ signal to the 82586 after power up. The LANHIB powers up the 82501 in non-loopback mode.

After making sure that the 82501 is generating proper signals to the 82586, the TSMS program is restarted with an initialization shown in Figure 20. The 82586 is configured to EXT-LPBK = 1, TONO-CRS = 1, and MIN-FRM-LEN = 6. The chip is also loaded with a destination address identical to the source address. If there are no problems in the 82501/82502 circuits, the station will be receiving its own transmitted frames. If problems exist, the station will only be transmitting. Since the 82586 is configured to TONO-CRS (Transmission On NO Carrier Sense), the chip will keep transmitting regardless of the state of carrier sense. The 82501/82502 circuits can then be probed with an oscilloscope at the locations indicated in Figure 21. Probing will catch problems like wiring mistakes, missing load resistors, etc.

Once the station is debugged, it can be connected to the network. If there is a problem in the network, the 82586's TDR command can be used to find the location and nature of the problem.

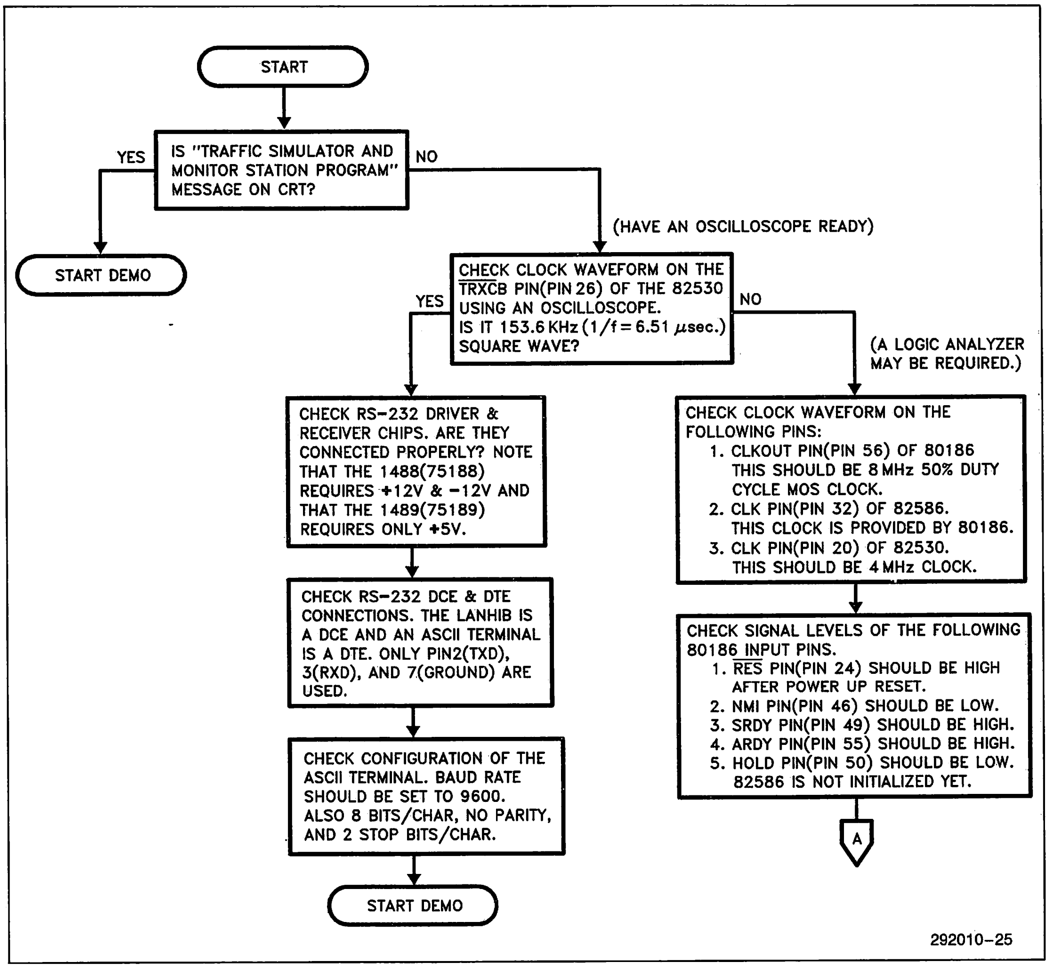
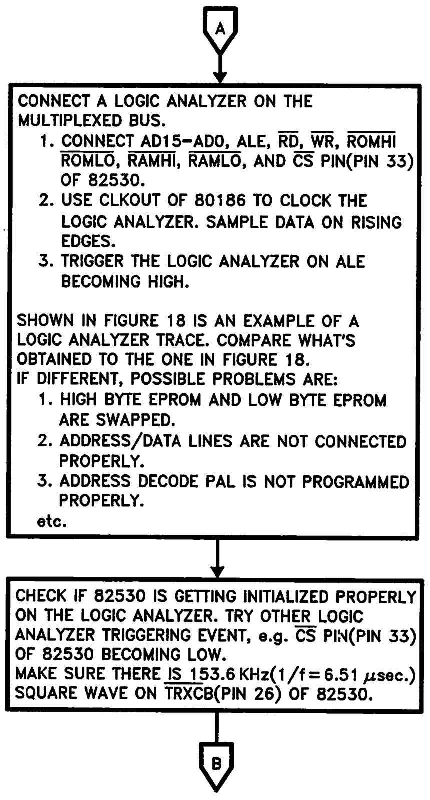

START
IS "TRAFFIC SIMULATOR AND MONITOR STATION PROGRAM" MESSAGE ON CRT?
YES
NO
START DEMO
(HAVE AN OSCILLOSCOPE READY)
CHECK CLOCK WAVEFORM ON THE TRXCB PIN(PIN 26) OF THE 82530 USING AN OSCILLOSCOPE. IS IT 153.6 KHz (1/f = 6.51 μsec.) SQUARE WAVE?
YES
NO
(A LOGIC ANALYZER MAY BE REQUIRED.)
CHECK RS-232 DRIVER & RECEIVER CHIPS. ARE THEY CONNECTED PROPERLY? NOTE THAT THE 1488(75188) REQUIRES +12V & −12V AND THAT THE 1489(75189) REQUIRES ONLY +5V.
CHECK RS-232 DCE & DTE CONNECTIONS. THE LANHIB IS A DCE AND AN ASCII TERMINAL IS A DTE. ONLY PIN2(TXD), 3(RXD), AND 7(GROUND) ARE USED.
CHECK CONFIGURATION OF THE ASCII TERMINAL. BAUD RATE SHOULD BE SET TO 9600. ALSO 8 BITS/CHAR, NO PARITY, AND 2 STOP BITS/CHAR.
START DEMO
CHECK CLOCK WAVEFORM ON THE FOLLOWING PINS:
1. CLKOUT PIN(PIN 56) OF 80186 THIS SHOULD BE 8 MHz 50% DUTY CYCLE MOS CLOCK.
2. CLK PIN(PIN 32) OF 82586. THIS CLOCK IS PROVIDED BY 80186.
3. CLK PIN(PIN 20) OF 82530. THIS SHOULD BE 4 MHz CLOCK.
CHECK SIGNAL LEVELS OF THE FOLLOWING 80186 INPUT PINS.
1. RES PIN(PIN 24) SHOULD BE HIGH AFTER POWER UP RESET.
2. NMI PIN(PIN 46) SHOULD BE LOW.
3. SRDY PIN(PIN 49) SHOULD BE HIGH.
4. ARDY PIN(PIN 55) SHOULD BE HIGH.
5. HOLD PIN(PIN 50) SHOULD BE LOW. 82586 IS NOT INITIALIZED YET.
A
292010–25

Figure 17. Flowchart for 80186/82586 System Troubleshooting



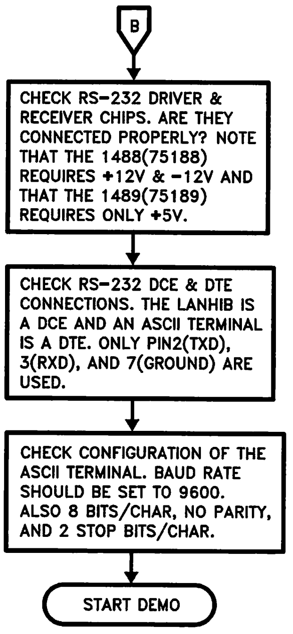
CONNECT A LOGIC ANALYZER ON THE MULTIPLEXED BUS.
1. CONNECT AD15–AD0, ALE, $\overline{RD}$, $\overline{WR}$, $\overline{ROMHI}$ $\overline{ROMLO}$, $\overline{RAMHI}$, $\overline{RAMLO}$, AND $\overline{CS}$ PIN(PIN 33) OF 82530.
2. USE CLKOUT OF 80186 TO CLOCK THE LOGIC ANALYZER. SAMPLE DATA ON RISING EDGES.
3. TRIGGER THE LOGIC ANALYZER ON ALE BECOMING HIGH.

SHOWN IN FIGURE 18 IS AN EXAMPLE OF A LOGIC ANALYZER TRACE. COMPARE WHAT'S OBTAINED TO THE ONE IN FIGURE 18.
IF DIFFERENT, POSSIBLE PROBLEMS ARE:
1. HIGH BYTE EPROM AND LOW BYTE EPROM ARE SWAPPED.
2. ADDRESS/DATA LINES ARE NOT CONNECTED PROPERLY.
3. ADDRESS DECODE PAL IS NOT PROGRAMMED PROPERLY.

etc.

CHECK IF 82530 IS GETTING INITIALIZED PROPERLY ON THE LOGIC ANALYZER. TRY OTHER LOGIC ANALYZER TRIGGERING EVENT, e.g. $\overline{CS}$ PIN(PIN 33) OF 82530 BECOMING LOW.
MAKE SURE THERE IS 153.6 KHz($1/f = 6.51$ μsec.) SQUARE WAVE ON $\overline{TRXCB}$(PIN 26) OF 82530.

292010–26



CHECK RS-232 DRIVER & RECEIVER CHIPS. ARE THEY CONNECTED PROPERLY? NOTE THAT THE 1488(75188) REQUIRES +12V & −12V AND THAT THE 1489(75189) REQUIRES ONLY +5V.

CHECK RS-232 DCE & DTE CONNECTIONS. THE LANHIB IS A DCE AND AN ASCII TERMINAL IS A DTE. ONLY PIN2(TXD), 3(RXD), AND 7(GROUND) ARE USED.

CHECK CONFIGURATION OF THE ASCII TERMINAL. BAUD RATE SHOULD BE SET TO 9600. ALSO 8 BITS/CHAR, NO PARITY, AND 2 STOP BITS/CHAR.

292010–27

Figure 17. Flowchart for 80186/82586 System Troubleshooting (Continued)

```
                              AD15-AD0
                              ALE
                              RD#
                              WR#
                              ROMHI#  ]
                              ROMLO#  } THESE ARE MEMORY CHIP SELECTS
                              RAMHI#  }
                              RAMLO#  ]
                              CS# PIN (PIN 33) OF 82530

0097 00 41 01001111
0098 00 41 01001111
0099 00 41 01101111
TRIG 00 41 11101111  ..... LOGIC ANALYZER IS TRIGGERED ON ALE=HI.
0101 FF F0 01001111  ..... 80186 JUMPS TO FFF0H AFTER RESET.
0102 06 EA 00101111  ..... JMP INSTRUCTION (DIRECT INTERSEGMENT)
0103 06 EA 00101111        SEGMENT OFFSET=0006H
0104 06 EA 00101111        SEGMENT SELECTOR=FFC0H
0105 06 EA 00101111        (80186 INSERTS 3 WAIT STATES BEFORE
0106 06 EA 00101111         UMCS REGISTER IS PROGRAMMED.)
0107 06 EA 11101111
0108 FF F2 01101111
0109 C0 40 00101111
0110 C0 00 00101111
0111 C0 00 00101111
0112 C0 00 00101111
0113 C0 00 00101111
0114 C0 00 11101111
0115 FF F4 01101111
0116 FF FF 00101111
0117 FF FF 00101111
0118 FF FF 00101111
0119 FF FF 00101111
0120 FF FF 00101111
0121 FF FF 11101111
0122 FF F6 01101111
0123 00 40 00101111
0124 00 00 00101111
0125 00 00 00101111
0126 00 00 00101111
0127 00 00 00101111
0128 00 00 11101111
0129 FC 06 01101111  ..... JUMPED TO FC06H
0130 2E FA 00101111
0131 2E FA 00101111
0132 2E FA 00101111
0133 2E FA 00101111
0134 2E FA 00101111
0135 2E FA 11101111
0136 FC 08 01101111
0137 16 8E 00101111
0138 16 8E 00101111
```

292010-28

Figure 18. Example of Logic Analyzer Trace

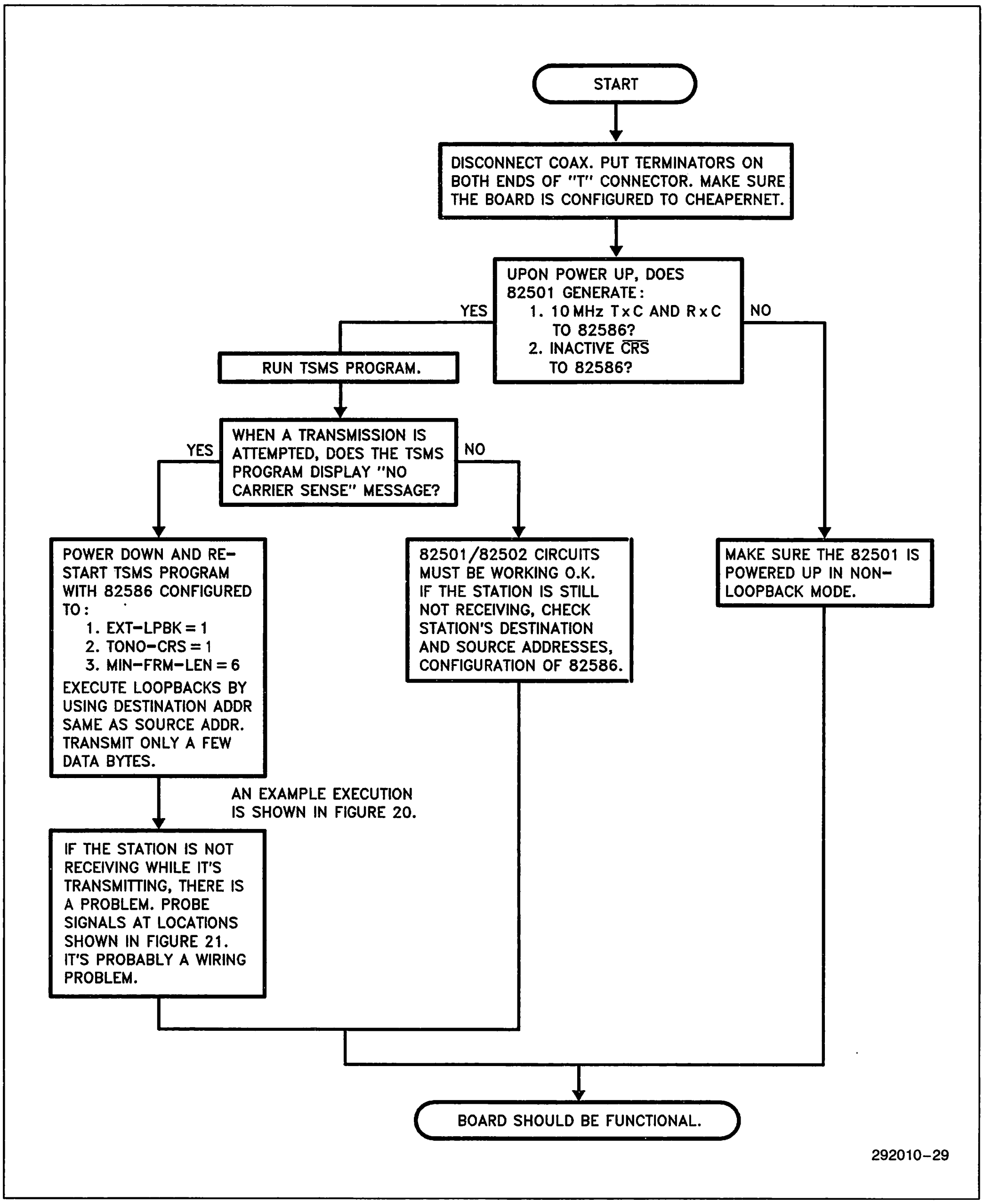

START
DISCONNECT COAX. PUT TERMINATORS ON BOTH ENDS OF "T" CONNECTOR. MAKE SURE THE BOARD IS CONFIGURED TO CHEAPERNET.
UPON POWER UP, DOES 82501 GENERATE: 1. 10 MHz TxC AND RxC TO 82586? 2. INACTIVE $\overline{CRS}$ TO 82586?
YES
NO
RUN TSMS PROGRAM.
WHEN A TRANSMISSION IS ATTEMPTED, DOES THE TSMS PROGRAM DISPLAY "NO CARRIER SENSE" MESSAGE?
YES
NO
POWER DOWN AND RE-START TSMS PROGRAM WITH 82586 CONFIGURED TO: 1. EXT-LPBK = 1 2. TONO-CRS = 1 3. MIN-FRM-LEN = 6 EXECUTE LOOPBACKS BY USING DESTINATION ADDR SAME AS SOURCE ADDR. TRANSMIT ONLY A FEW DATA BYTES.
AN EXAMPLE EXECUTION IS SHOWN IN FIGURE 20.
IF THE STATION IS NOT RECEIVING WHILE IT'S TRANSMITTING, THERE IS A PROBLEM. PROBE SIGNALS AT LOCATIONS SHOWN IN FIGURE 21. IT'S PROBABLY A WIRING PROBLEM.
82501/82502 CIRCUITS MUST BE WORKING O.K. IF THE STATION IS STILL NOT RECEIVING, CHECK STATION'S DESTINATION AND SOURCE ADDRESSES, CONFIGURATION OF 82586.
MAKE SURE THE 82501 IS POWERED UP IN NON-LOOPBACK MODE.
BOARD SHOULD BE FUNCTIONAL.
292010-29

Figure 19. Flowchart for 82501/82502 Circuits Troubleshooting

Traffic Simulator and Monitor Station Program

```
Initialization begun

Configure command is set up for default values.
Do you want to change any bytes? (Y or N) ==> Y
Enter byte number (1 - 11) ==> 4
Enter byte 4 (4H) ==> A6H
Any more bytes? (Y or N) ==> Y
Enter byte number (1 - 11) ==> 9
Enter byte 9 (9H) ==> 08H
Any more bytes? (Y or N) ==> Y
Enter byte number (1 - 11) ==> 11
Enter byte 11 (BH) ==> 6
Any more bytes? (Y or N) ==> N
Configure the 586 with the prewired board address ==> N
Enter this station's address in Hex ==> 000000002200
You can enter up to 8 Multicast Addresses.
Would you like to enter a Multicast Address? (Y or N) ==> N
You entered 0 Multicast Address(es).

Would you like to transmit?
Enter a Y or N ==> Y
Enter a destination address in Hex ==> 000000002200

Enter TYPE ==> 0
How many bytes of transmit data?
Enter a number ==> 2
Transmit Data is continuous numbers (0, 1, 2, 3, ... )
Change any data bytes? (Y or N) ==> N
Enter a delay count ==> 0
Setup a transmit terminal count? (Y or N) ==> N

Destination Address: 00  00  00  00  22  00
Frame Length: 20 bytes
Time Interval between Transmit Frames: 159.4 seconds
Network Percent Load generated by this station: 11.0 %
Transmit Frame Terminal Count: Not Defined

Good enough? (Y or N) ==> Y
```

292010-77

Figure 20. TSMS Initialization for 82501/82502 Circuits Troubleshooting

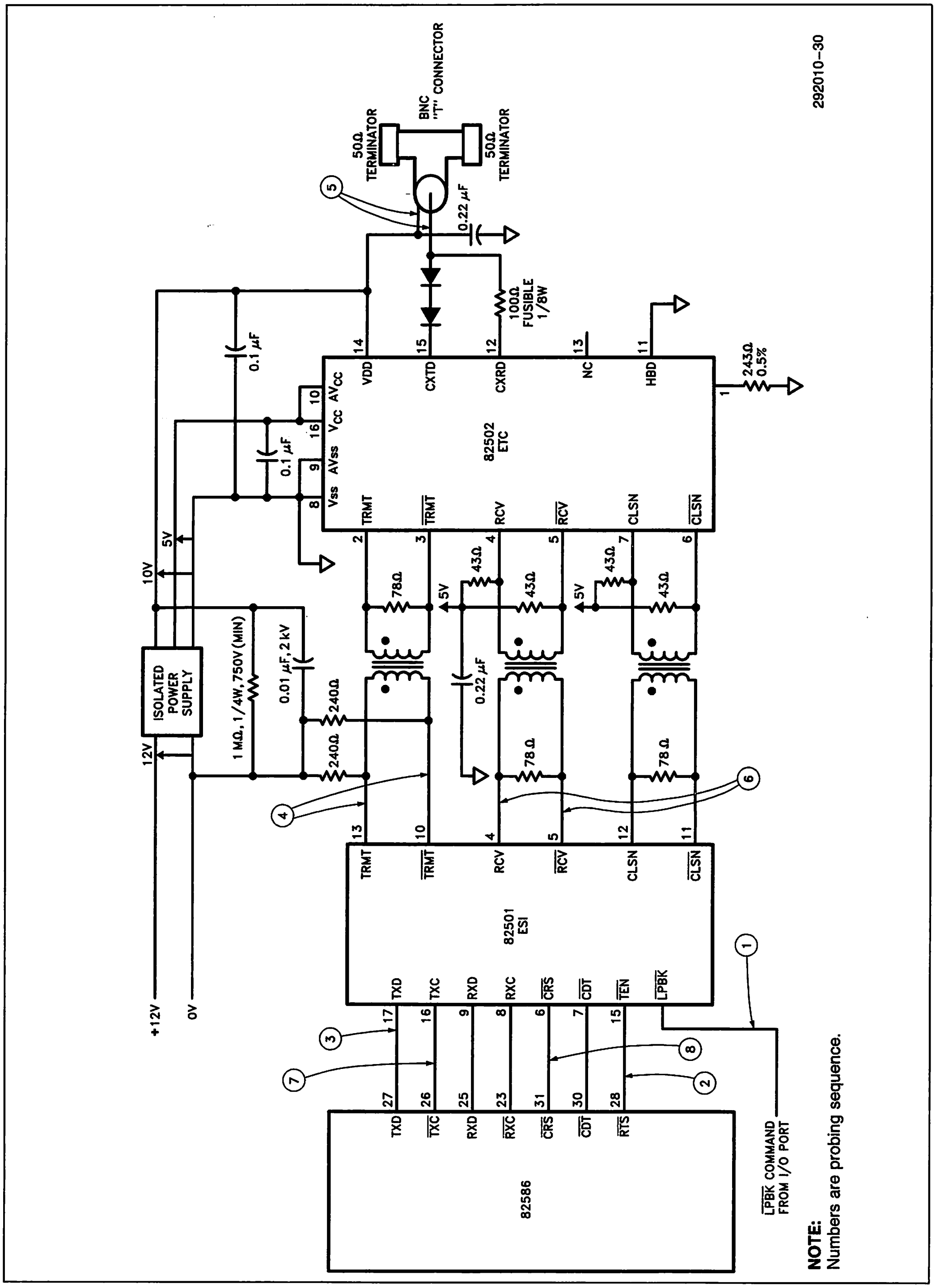
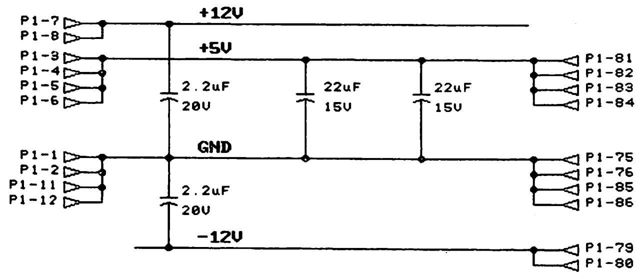


Figure 21. Probing 82501/82502 Circuits

APPENDIX A
LANHIB SCHEMATICS
PARTS LIST
PAL EQUATIONS
DIP SWITCH SETTINGS
WIRE WRAP SERVICES

PARTS LIST

REFERENCES	DESCRIPTION	MFR. PART NO.	MFR. CODE	QTY.
U1	IC	74S32	OBD	1
U2	IC	74LS04	OBD	1
U3, U4	IC	74LS245	OBD	2
U5	IC	74F373	OBD	1
U6, U7	IC	74LS373	OBD	2
U8	IC	80186	INT	1
U9	IC	82586	INT	1
U10	IC	16L8	OBD	1
U11	IC	74LS02	OBD	1
U12, U27	IC	74LS74	OBD	2
U28	IC	74AS74	OBD	1
U13	IC	74LS165	OBD	1
U14	IC	82501	INT	1
U15	Pulse Transformer Pack	PE64102	PE	1
U16	IC	82502	INT	1
U17	DC/DC Converter	PE64369	PE	1
U18, U20	IC, 64K-Bit EPROM	2764-20	INT	2
U26	IC	74AS08	OBD	1
U22, U23	IC, SRAM	HM6264-15	HIT	2
U24	IC, 256-Bit PROM	TBP18S030	TI	1
U25	IC	74AS04	OBD	1
U29	IC	82530	INT	1
U30	IC	1489	OBD	1
U31	IC	1488	OBD	1
U32	IC, 1M-Bit EPROM (Optional)	27210	INT	1
R1-R3, R6 R19, R20	Resistor, 10K ohm, 1/4W, 5%	COML	OBD	6
R4	Resistor, 100K ohm, 1/4W, 5%	COML	OBD	1
R5	Resistor, 2.2K ohm, 1/4W, 5%	COML	OBD	1
R7, R8, R12	Resistor, 78.7 ohm, 1/8W, 1%	COML	OBD	3
R9, R10	Resistor, 240 ohm, 1/4W, 5%	COML	OBD	2
R11	Resistor, 1M ohm, 1/4W, 750Vdc (min), 5%	COML	OBD	1
R13-R16	Resistor, 43.2ohm 1/8W, 1%	COML	OBD	4
R17	Resistor, 100 ohm, Fusible 1/8W, 5%	COML	RCD	1
R18	Resistor, 243 ohm, 1/8W, 0.5%	COML	OBD	1
R21, R22	Resistor, 5K ohm, 1/4W, 5%	COML	OBD	2
RP1	Resistor Pack, 1K ohm, 16 pin	COML	OBD	1
R23-R26	Resistor, 1K ohm, 1/4W, 5%	COML	OBD	4
C1, C2	Capacitor, 20pF, 100V, 5%	COML	OBD	2
C3	Capacitor, 10uF, 20V	COML	OBD	1
C4, C5	Capacitor, 30pF, 100V, 5%	COML	OBD	2
C6	Capacitor, 0.022uF, 50V	COML	OBD	1
C7	Capacitor, 1.0uF, 50V	COML	OBD	1
C11, C12	Capacitor, 0.01uF, 50V	COML	OBD	2
C10	Capacitor, 0.01uF, 2KV	COML	OBD	1
C8, C9	Capacitor, 0.22uF, 50V	COML	OBD	2
CR1	Diode	1N914	OBD	1
CR2, CR3	Diode	1N5282	OBD	2
Y1	Parallel Resonant Crystal, 16M Hz	COML	OBD	1
Y2	Parallel Resonant Crystal, 20M Hz	COML	OBD	1



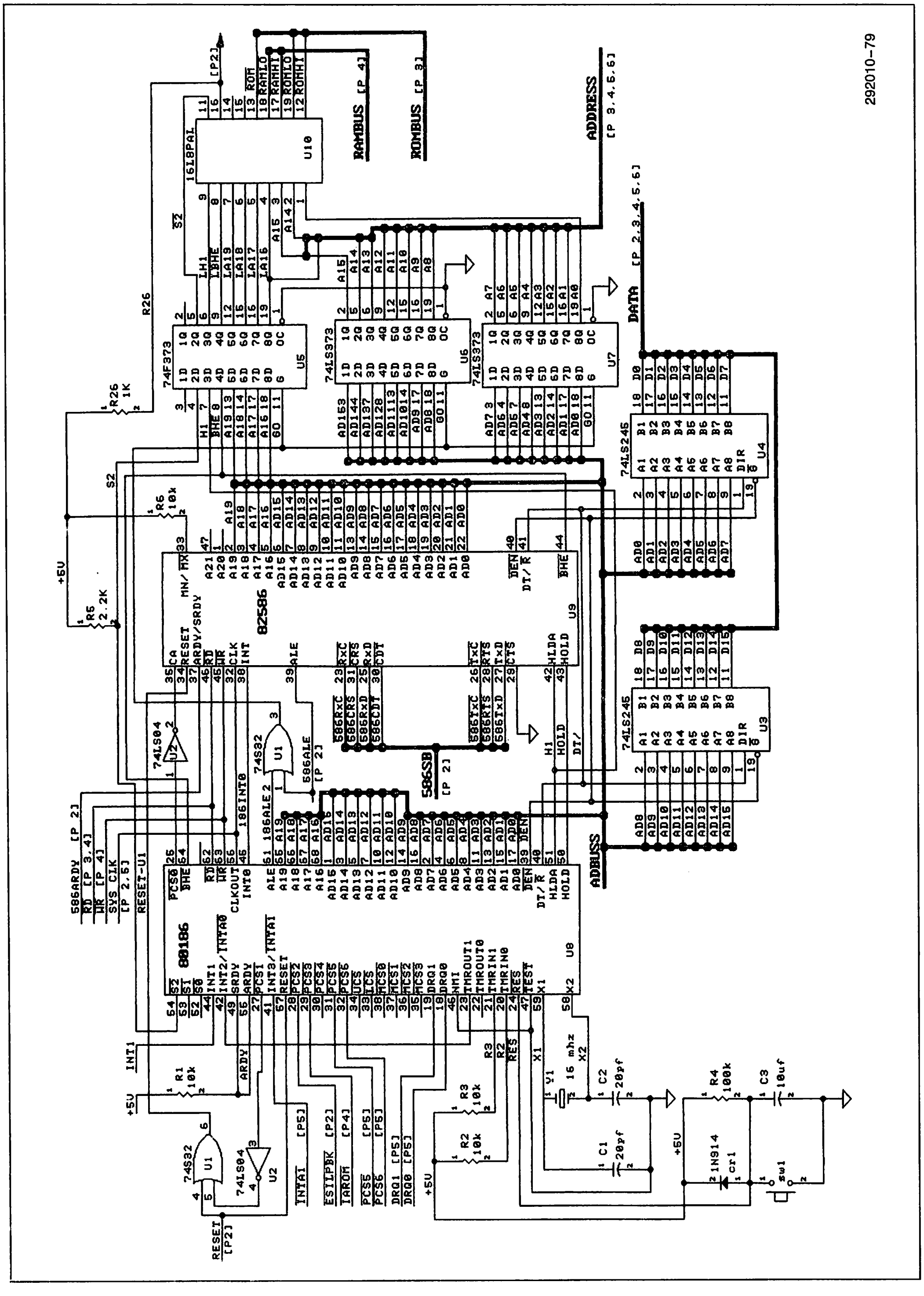
POWER SUPPLY CONNECTIONS

NOTES:

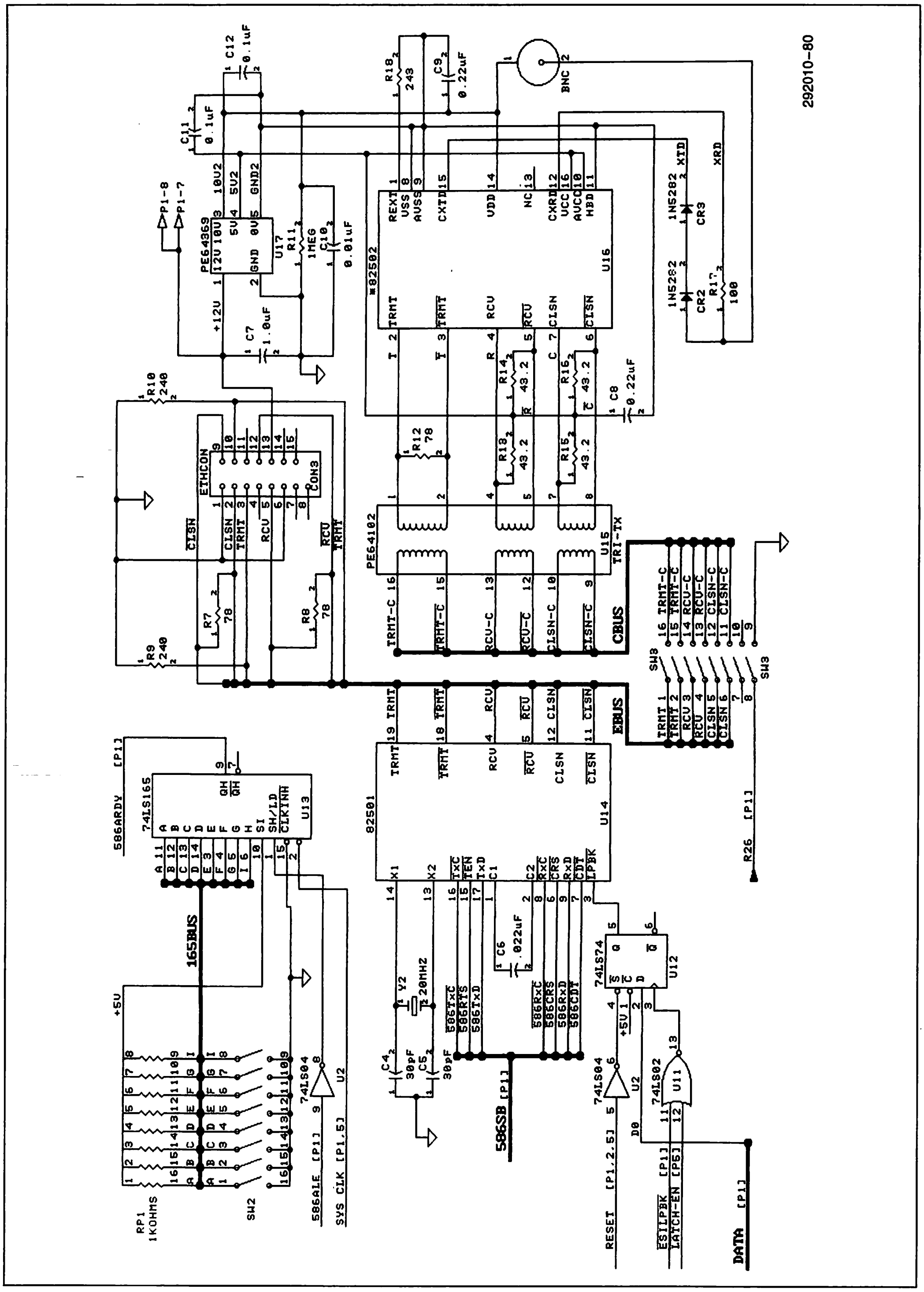
1. THE BOARD REQUIRES +5V, +12V, AND -12V. MULTIBUS POWER PINS FOR THESE VOLTAGES AND GROUND ARE SHOWN ABOVE.
2. EACH IC SHOULD HAVE A 0.1uF CAPACITOR BETWEEN POWER PIN AND GROUND PIN. PARTS LIST DOES NOT INCLUDE DECOUPLING CAPACITORS.
3.

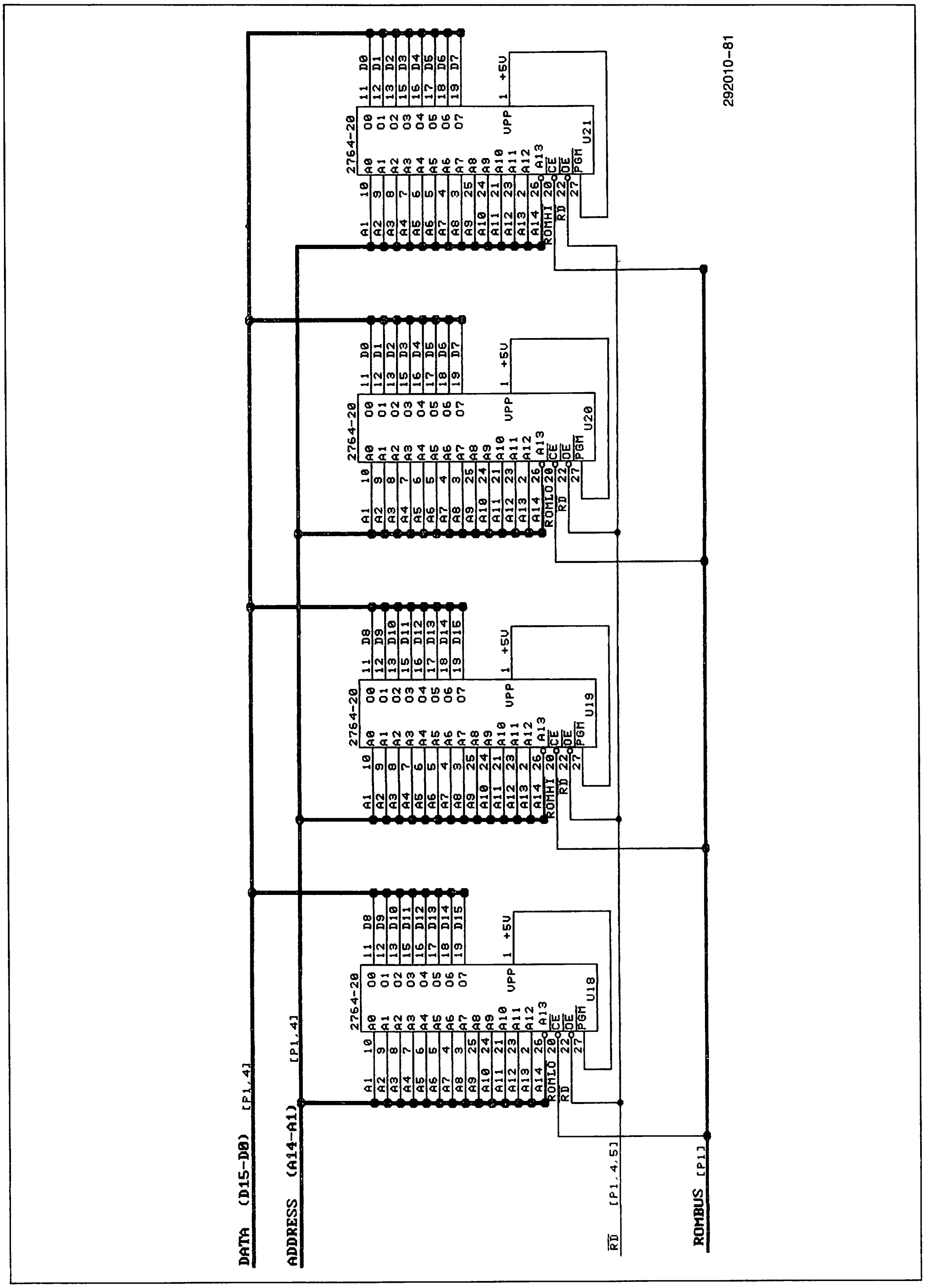
MTR. CODE	MANUFACTURE	LOCATION
INT	INTEL CORPORATION	SANTA CLARA, CA
HIT	HITACH AMERICA LTD.	SAN JOSE, CA
OBD	ORDER BY DESCRIPTION (ANY COMMERCIAL (COML) SOURCE)	
PE	PULSE ENGINEERING	SAN DIEGO, CA
RCD	RCD COMPONENTS INC.	MANCHESTER, NH
TI	TEXAS INSTRUMENTS	DALLAS, TX

292010-78

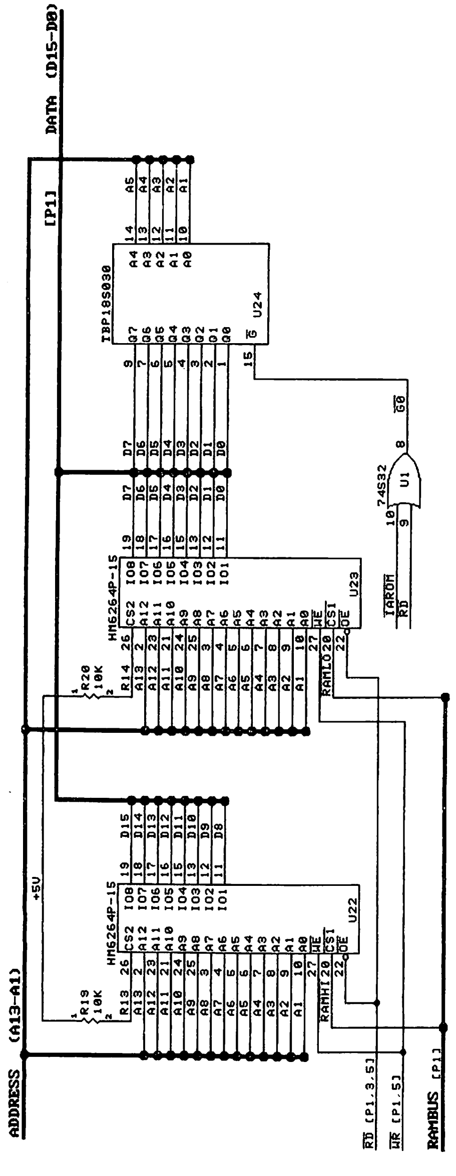

80186
U8
82586
U9
74F373
U5
74LS373
U6
74LS373
U7
74LS245
U4
74LS245
U3
16L8PAL
U10
74LS04
U2
74S32
U1
RAMBUS [P 4]
ROMBUS [P 3]
ADDRESS [P 3,4,5,6]
DATA [P 2,3,4,5,6]
ADBUSS
586SB [P 2]
586ALE [P 2]
586ARDY [P 2]
RD [P 3,4]
WR [P 4]
SYS CLK [P 2,5]
RESET-U1
RESET [P2]
INTA1 [P5]
ESILPBK [P2]
IAROM [P4]
PCS5 [P5]
PCS6 [P5]
DRQ1 [P5]
DRQ0 [P5]
+5V
R1 10k
R2 10k
R3 10k
R4 100k
R5 2.2K
R6 10k
R26 1K
C1 20pf
C2 20pf
C3 10uf
Y1 16 mhz
1N914 cr1
sw1

292010-79


292010-80
82501
U14
82502
U16
PE64102
U15
TRI-TX
PE64369
U17
74LS165
U13
74LS74
U12
74LS02
U11
74LS04
U2
ETHCON
CON3
BNC
CBUS
EBUS
165BUS
586SB
DATA
SW2
SW3
RP1
1KOHMS
+5V
+12V
Y2
20MHZ
1N5282
CR2
CR3
XTD
XRD
586ARDY [P1]
586ALE [P1]
SYS CLK [P1.5]
RESET [P1.2.5]
ESILPBK [P1]
LATCH-EN [P5]
R26 [P1]
P1-8
P1-7

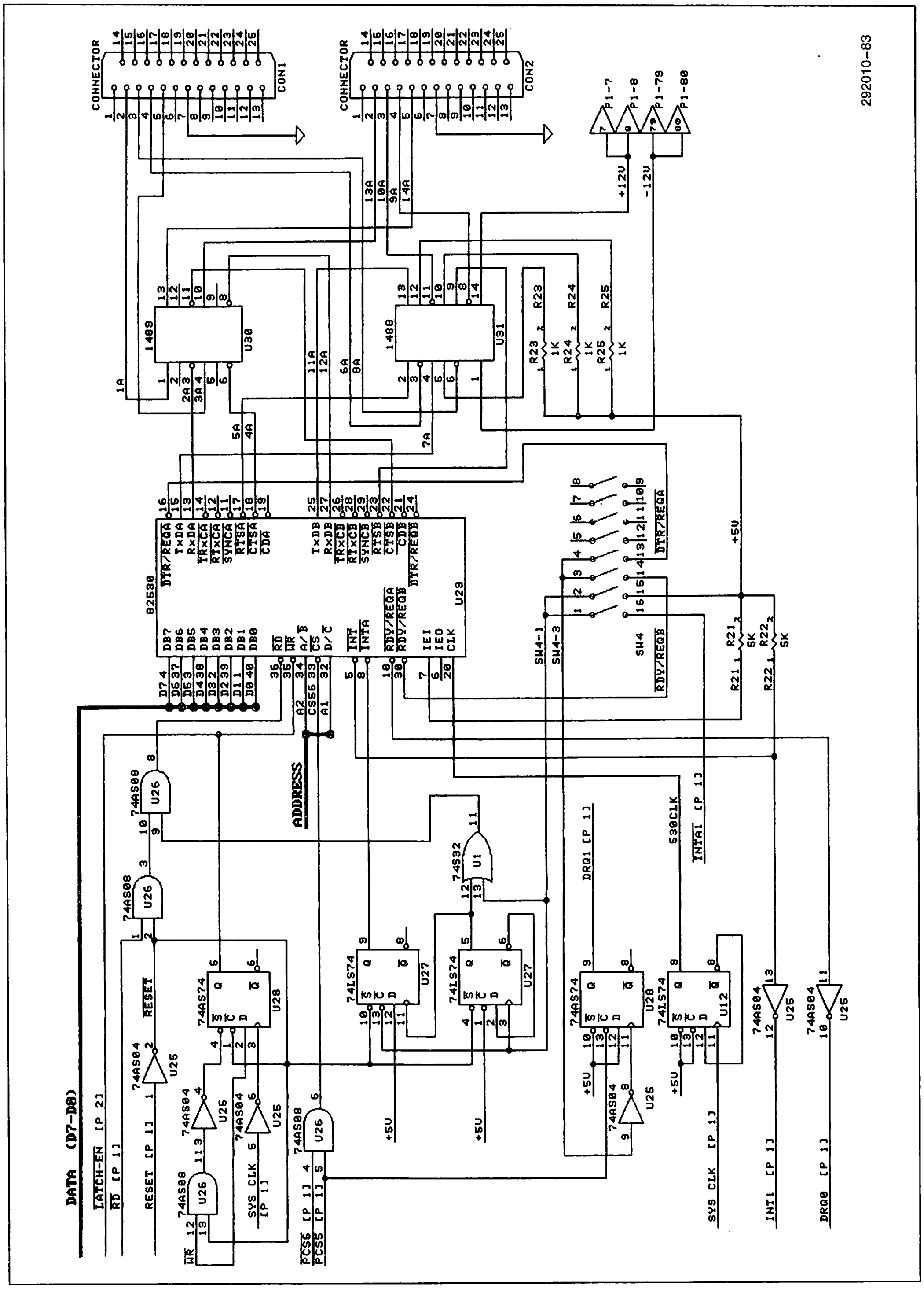

DATA (D15-D0) [P1,4]
ADDRESS (A14-A1) [P1,4]
RD [P1,4,5]
ROMBUS [P1]
2764-20
U18
U19
U20
U21
ROMLO
ROMHI
RD
+5V
VPP
CE
OE
PGM
A0
A1
A2
A3
A4
A5
A6
A7
A8
A9
A10
A11
A12
A13
A14
O0
O1
O2
O3
O4
O5
O6
O7
D0
D1
D2
D3
D4
D5
D6
D7
D8
D9
D10
D11
D12
D13
D14
D15

292010-81

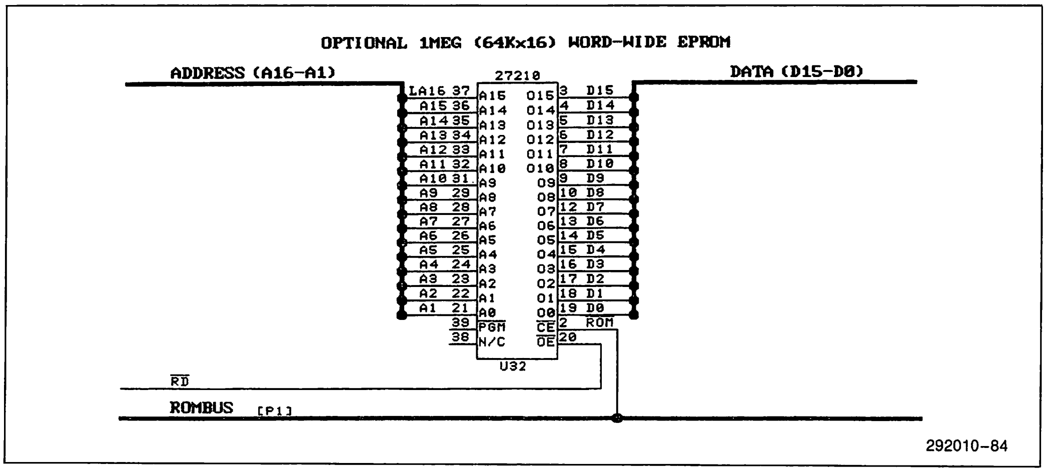


DATA (D15-D0)
[P1]
ADDRESS (A13-A1)
TBP18S030
U24
HM6264P-15
U23
U22
74S32
U1
G0
IAROM
RD
RAMLO
RAMHI
R19 10K
R20 10K
+5V
RD [P1.3.5]
WR [P1.5]
RAMBUS [P1]

292010-82


CONNECTOR
CON1
CON2
P1-7
P1-8
P1-79
P1-80
+12V
-12V
1489
U30
1488
U31
R23 1K
R24 1K
R25 1K
82530
U29
DTR/REQA
TxDA
RxDA
TRxCA
RTxCA
SYNCA
RTSA
CTSA
CDA
TxDB
RxDB
TRxCB
RTxCB
SYNCB
RTSB
CTSB
CDB
DTR/REQB
DB7
DB6
DB5
DB4
DB3
DB2
DB1
DB0
RD
WR
A/B
CS
D/C
INT
INTA
RDY/REQA
RDY/REQB
IEI
IEO
CLK
SW4
SW4-1
SW4-3
R21 5K
R22 5K
+5V
DATA (D7-D0)
LATCH-EN [P 2]
RD [P 1]
RESET [P 1]
WR
SYS CLK [P 1]
PCS6 [P 1]
PCS5 [P 1]
ADDRESS
A2
CS56
A1
74AS08
74AS04
74AS74
74LS74
74S32
U1
U25
U26
U27
U28
U12
RESET
DRQ1 [P 1]
530CLK
INTA1 [P 1]
INT1 [P 1]
DRQ0 [P 1]

292010-83



```
Module Addr_dec
Title 'LANHIB Address Decode Logic
       Kiyoshi Nishide   Intel Corp.   March, 1986'

"Declarations

    PAL1                            device            'P16L8';

    A0, A14, A15        pin                           1, 2, 3;
    A16, A17, A18       pin                           4, 5, 6;
    A19, BHE                        pin               7, 8;
    HLDA, S2                        pin               9, 11;
    RAMLO, RAMHI        pin                           18, 17;
    ROMLO, ROMHI        pin                           19, 12;
    ROM                 pin         13;
    R104                pin         16;

Equations

    !ROMHI  =  A15 & A16 & A17 & A18 & A19 & (HLDA # S2) & R104;
    !ROMLO  =  !A15 & A16 & A17 & A18 & A19 & (HLDA # S2) & R104;
    !ROM  =  A17 & A18 & A19 & (HLDA # S2) & !R104;
    !RAMHI  =  !A14 & !A15 & !A16 & !A17 & !A18 & !A19 & !BHE & (HLDA # S2);
    !RAMLO  =  !A0 & !A14 & !A15 & !A16 & !A17 & !A18 & !A19 & (HLDA # S2);

End Addr_dec
```

PAL Equations

DIP SWITCH SETTINGS FOR VARIOUS OPERATIONS

"1" indicates ON (Switch is closed).
"0" indicates OFF (Switch is open).
"X" indicates Don't Care.

1. To configure the board to Ethernet or Cheapernet:

	SW3 87654321	Comment
Ethernet	XX000000	
Cheapernet	XX111111	Transceiver Cable should not be connected.

2. To run the TSMS program or the Data Link Driver program:

	SW4 87654321	Comment
TSMS Program or Data Link Driver Program	XXXX0001	TSMS program uses the 82530 in Asynchronous Polling mode. Data Link Driver program uses the 825830 in Asynchronous Polling and Vectored Interrupt modes.

3. To select the 2764-20 EPROMs or 27210 EPROM:

	SW3 87654321
2764-20 EPROMs	1XXXXXXX
27210 EPROM	0XXXXXXX

4. Dip Switch Setting Examples:

	SW3 87654321	SW4 87654321
1) To run the TSMS Program from the 2764-20 EPROMs in Cheapernet Configuration	1X111111	XXXX0010
2) To run the TSMS Program from the 2764-20 EPROMs in Ethernet Configuration	1X000000	XXXX0010
3) To run the TSMS Program or the Data Link Driver program from the 27210 EPROM in Cheapernet Configuration	0X111111	XXXX0001
4) To run the TSMS Program or the Data Link Driver program from the 27210 EPROM in Ethernet Configuration	0X000000	XXXX0001

5. Dip Switch SW2 programs the number of wait states for the 82586 (see Table 3).

WIRE WRAP SERVICES SUPPORTING WRAPID

AUGAT***
Interconnection Systems Division

40 Perry Avenue
P.O. Box 1037
Attleboro, MA 02703
(617) 222-2202

100935 South Wilcrest Drive
Houston, TX 77099
(713) 495-3100

Automation Delectronics Corporation

1650 Locust Avenue
Bohemia, NY 11716
(516) 567-7007

dataCon, Inc.***

Eastern Division
60 Blanchard Road
Burlington, MA 01803
(617) 273-5800

Mid-Western Division
502 Morse Avenue
Schaumburg, IL 60193
(312) 529-7690

Western Division
20150 Sunburst Street
Chatsworth, CA 91311-6280
(818) 700-0600

South-Western Division
1829 Monetary Lane
Carrollton, TX 75006
(214) 245-6161

European Division
In der Klinge 5
D-7100 Heilbronn, West Germany
(01731) 217 12

DATAWRAP

37 Water Street
Wakefield, MA 01880
(617) 938-8911

Elma/EMS
A Division of Sandberg Industries

Berkshire Industrial Park
Bethel, CT 06801
(203) 797-9711

1851 Reynolds Avenue
Irvine, CA 92714
(714) 261-9473

3042 Scott Boulevard
Santa Clara, CA 95054
(408) 970-8874

WRAPEX Corporation

96 Mill Street
Woonsocket, RI 02895
(401) 769-3805

November 1985
*** WRAPID distributors

NOTE:
If your wire wrap service company is not listed here have them contact COMPION, Inc. WRAPID output specifications are provided free of charge to any interested wire wrap services.

APPENDIX B SOFTWARE LISTINGS—TSMS PROGRAM AND LANHIB INITIALIZATION ROUTINE

```
/****************************************************************************/
/*                                                                          */
/*              Traffic Simulator/Monitor Station Program                   */
/*              for 186/586 High Integration Board and                      */
/*              iSBC 186/51                                                 */
/*                                                                          */
/*              Ver. 1.0              December 17, 1984                     */
/*                                                                          */
/*              Kiyoshi Nishide       Intel Corporation                     */
/*                                                                          */
/****************************************************************************/

/* This software can be conditionally compiled to work on the iSBC 186/51 or
   on the LANHIB.  If 'set(SBC18651)' is added to the compiler call statement,
   this source program will be compiled for the iSBC18651.  */

tsms:

do;

declare main label public;

/* literals */

$IF SBC18651

declare lit             literally 'literally',
        true                  lit '1',
        false                 lit '0',
        forever               lit 'while 1',
        ISCP$LOC$LO           lit '0FFF0H',
        ISCP$LOC$HI           lit '0',
        SCB$BASE$LO           lit '0',
        SCB$BASE$HI           lit '0',
        CA$PORT               lit '0C8H',
        BOARD$ADDRESS$BASE    lit '0F0H',
        INT$TYPE$586          lit '20H',
        INT$TYPE$TIMER0       lit '30H',
        INT$CTL$TIMER0        lit '0FF32H',
        INT$7                 lit '27H',
        PIC$MASK$130          lit '0E2H',
        PIC$MASK$186          lit '0FF28H',
        ENABLE$586            lit '0FEH',
        ENABLE$586$186        lit '0EEH',
        PIC$EOI$130           lit '0E0H',
        EOI$CMD0$130          lit '60H',
        EOI$CMD4$130          lit '64H',
        PIC$EOI$186           lit '0FF22H',
        EOI$CMD0$186          lit '0',
        PIC$VTR$186           lit '0FF20H',
```

292010-31

Traffic Simulator/Monitor Station Program

```
                TIMERO$CTL          lit 'OFF56H',
                TIMERO$COUNT        lit 'OFF50H',
                MAX$COUNT$A         lit 'OFF52H',
                CA                  lit '0',
                ESI$PORT            lit '0CBH',
                NO$LOOPBACK         lit '8',
                LOOPBACK            lit '0';

        $ELSE

3   1   declare lit              literally 'literally',
                true                lit '1',
                false               lit '0',
                forever             lit 'while 1',
                ISCP$LOC$LO         lit '03FF8H',
                ISCP$LOC$HI         lit '0',
                SCB$BASE$LO         lit '0',
                SCB$BASE$HI         lit '0',
                CA$PORT             lit '8000H',
                BOARD$ADDRESS$BASE  lit '8180H',
                INT$TYPE$586        lit '12',
                INT$TYPE$TIMERO     lit '8',
                INT$CTL$TIMERO      lit 'OFF32H',
                PIC$MASK$186        lit 'OFF28H',
                ENABLE$586          lit '0EFH',
                ENABLE$586$186      lit '0EEH',
                PIC$EOI$186         lit 'OFF22H',
                EOI$CMD0$186        lit '12',
                EOI$CMD4$186        lit '8',
                TIMERO$CTL          lit 'OFF56H',
                TIMERO$COUNT        lit 'OFF50H',
                MAX$COUNT$A         lit 'OFF52H',
                CA                  lit '0',
                ESI$PORT            lit '8100H',
                NO$LOOPBACK         lit '1',
                LOOPBACK            lit '0';

        $ENDIF

        $IF NOT SBC18651

        /* System Configuration Pointer */

4   1   declare scp structure
                (
                sysbus byte,
                unused (5) byte,
                iscp$addr$lo word,
                iscp$addr$hi word
                )
                at (OFFFF6H) data (0, 0, 0, 0, 0, 0, ISCP$LOC$LO, ISCP$LOC$HI);

        $ENDIF

        /* Intermediate System Configuration Pointer */
```

292010-32

Traffic Simulator/Monitor Station Program (Continued)

```
 5   1      declare iscp$ptr pointer,
                    iscp based iscp$ptr structure
                    (
                    busy byte,     /* set to 1 by CPU before its first CA to 586,
                                      cleared by 586 after reading info from it */
                    unused byte,   /* unused */
                    scb$o word,    /* offset of system control block */
                    scb$b (2) word /* base of system control block */
                    );

            /* System Control Block */

 6   1      declare scb structure
                    (
                    status word,        /* cause(s) of interrupt, CU state, RU state */
                    cmd word,           /* int acks, CU cmd, RESET bit, RU cmd */
                    cbl$offset word,    /* offset of first command block in CBL */
                    rpa$offset word,    /* offset of first packet descriptor in RPA */
                    crc$errs word,      /* crc error encounterd so far */
                    aln$errs word,      /* alignment errors */
                    rsc$errs word,      /* no resources */
                    ovrn$errs word      /* overrun errors */
                    );

            /* 82586 Action Commands */

            /* NOP */

 7   1      declare nop structure
                    (
                    status word,
                    cmd word,
                    link$offset word
                    );


            /* Individual Address Setup */

 8   1      declare ia$setup structure
                    (
                    status word,
                    cmd word,
                    link$offset word,
                    ia$address (6) byte
                    );

            /* Configure */

 9   1      declare configure structure
                    (
                    status word,
                    cmd word,
                    link$offset word,
                    byte$cnt byte,
                    info (11) byte
                    );
```

292010-33

Traffic Simulator/Monitor Station Program (Continued)

```
              /* Multicast Address Setup */

10    1       declare mc$setup structure
                      (
                      status word,
                      cmd word,
                      link$offset word,
                      mc$byte$count word,
                      mc$address (48) byte  /* only 8 MC addresses are allowed */
                      );


              /* Transmit */

              /* This transmit command is made of one transmit buffer descriptor and one
                 1518 bytes long buffer. */

11    1       declare transmit structure
                      (
                      status word,
                      cmd word,
                      link$offset word,
                      bd$offset word,
                      dest$adr (6) byte,
                      type word
                      );

              /* Transmit Buffer Descriptor */

12    1       declare tbd structure
                      (
                      act$count word,
                      link$offset word,
                      ad0 word,
                      ad1 word
                      );

              /* Transmit Buffer */

13    1       declare tx$buffer (1518) byte;


              /* TDR */

14    1       declare tdr structure
                      (
                      status word,
                      cmd word,
                      link$offset word,
                      result word
                      );

              /* Diagnose */

15    1       declare diagnose structure
                      (
```

292010-34

Traffic Simulator/Monitor Station Program (Continued)

```
                         status word,
                         cmd word,
                         link$offset word
                         );


                 /* Dump Status */

 16    1         declare dump structure
                         (
                         status word,
                         cmd word,
                         link$offset word,
                         buff$ptr word
                         );

                 /* Dump Area */

 17    1         declare dump$area (170) byte;


                 /* Frame Descriptor */

                 /* Receive frame area is made of 5 RFDs, 5 RBDs, and 5 1514 bytes long
                    buffers. */

 18    1         declare rfd (5) structure
                         (
                         status word,
                         el$s word,
                         link$offset word,
                         bd$offset word,
                         dest$adr (3) word,
                         src$adr (3) word,
                         type word
                         );

                 /* Receive Buffer Descriptor */

 19    1         declare rbd (5) structure
                         (
                         act$count word,
                         next$bd$link word,
                         ad0 word,
                         ad1 word,
                         size word
                         );

                 /* Receive Buffer */

 20    1         declare rbuf (5) structure
                         (buffer (1514) byte);


                 /* global variables */

 21    1         declare status word,            /* UART status */
```

292010-35

Traffic Simulator/Monitor Station Program (Continued)

```
                    actual word,                /* actual number of chars UART transferred */
                    c$buf (80) byte,            /* buffer for a line of chars */
                    dhex byte,                  /* number base switch */
                    ch byte at (@c$buf),
                    char$count byte,
                    receive$count dword,        /* counter for received frames */
                    count dword,                /* counter for transmitted frames */
                    preamble word,              /* preamble length in word */
                    address$length byte,        /* address length in byte */
                    ad$loc byte,                /* address location control of 82586 */
                    crc byte,                   /* crc length */
                    goback byte,                /* if set, go back to Continuous Mode */
                    reset byte,                 /* reset flag */
                    delay word,                 /* delay conunt for tranmission delay */
                    cur$cb$offset word,         /* offset of current command block */
                    current$frame byte,         /* offset of frame descriptor just used */
                    no$transmission byte,
                    stop$count dword,           /* transmit terminal frame count */
                    stop byte,
                    mc$count byte,
                    z byte,
                    y byte;

        /* external procedures */

        read: procedure (a, b, c, d, e) external;
        declare (a, c) word,
                (b, d, e) pointer;
        end read;


        write: procedure (a, b, c, d) external;
        declare (a, c) word,
                (b, d) pointer;
        end write;


        csts: procedure byte external;
        end csts;

        /* utility procedures */

        offset: procedure (ptr) word;

        /* This procedure takes a pointer variable (selector:offset), caluculates an
           absolute address, subtracts the 82586 SCB offset from the absolute address,
           and then returns the result as an offset value for the 82586  */

        declare (ptr, ptr$loc) pointer,
                base586 dword,
                w based ptr$loc (2) word;

            ptr$loc = @ptr;

            /* 82586 SCB Base Address (20-bit wide in this 186 based system) */
```

292010-36

Traffic Simulator/Monitor Station Program (Continued)

```
2          base586 = (shl(double (iscp.scb$b(1)), 16) and 000F0000H) + iscp.scb$b(0);
2          return low((shl(double (w(1)), 4) + w(0)) - base586);

2      end offset;


1      writeln: procedure (a, b, c, d);

       /* This procedure writes a line and put a CR/LF at the end. */

2      declare (a, c) word,
               (b, d) pointer;

2          call write(a, b, c, d);
2          call write(0, @(0DH, 0AH), 2, @status);

2      end writeln;


1      cr$lf: procedure;

       /* This procedure writes a CR/LF. */

2          call write (0, @(0DH, 0AH), 2, @status);

2      end cr$lf;


1      pause: procedure;

       /* This procedure breaks a program flow, and waits for a char to be typed. */

2          call write(0, @(0DH, 0AH, 'Hit <CR> to countinue'), 23, @status);
2          call read(1, @c$buf, 80, @actual, @status);
2          call cr$lf;

2      end pause;


1      skip: procedure byte;

       /* This procedure skips all leading blank characters and returns the first
          non-blank character.  */

2      declare i byte;

2          i = 0;
2          do while (c$buf(i) = ' ');
3              i = i + 1;
3          end;
2          return i;

2      end skip;


1      read$char: procedure byte;
```

292010-37

Traffic Simulator/Monitor Station Program (Continued)

```
              /* This procedure reads a line and returns ther first non-blank character.  */

58    2       declare i word;

59    2           call read(1, @c$buf, 80, @actual, @status);
60    2           i = skip;
61    2           return(c$buf(i));

62    2       end read$char;


63    1       read$bit: procedure byte;

              /* This procedure reads a bit and returns the value.  */

64    2       declare b byte;

65    2           do forever;
66    3               b = read$char;
67    3                   if b = '1' then return 1;
69    3               else
70    3                   if b = '0' then return 0;
71    3               else
                          call write(0, @(' Enter a 0 or 1 ==> '), 20, @status);
72    3           end;

73    2       end read$bit;


74    1       yes: procedure byte;

              /* This procedure reads a character and determines if it is a Y(y) or N(n).   */

75    2       declare b byte;

76    2           do forever;
77    3               b = read$char;
78    3               if (b = 'Y') or (b = 'y') then return true;
80    3               else
81    3                   if (b = 'N') or (b = 'n') then return false;
82    3               else
                          call write(0, @(0DH, 0AH, ' Enter a Y or N ==> '), 22, @status);
83    3           end;

84    2       end yes;


85    1       char$to$int: procedure (c) byte;

              /* This procedure converts a byte of ASCII integer to an integer.  */

86    2       declare c byte;

87    2           if ('0' <= c) and (c <= '9') then return (c - 30H);
89    2           else
90    2               if('A' <= c) and (c <= 'F') then return (c - 37H);
91    2               else
```

292010-38

Traffic Simulator/Monitor Station Program (Continued)

```
            if ('a' <= c) and (c <= 'f') then return (c - 57H);
            else return OFFH;

end char$to$int;


int$to$asci: procedure (value, base, ld, bufadr, ,width);

/* This procedure converts an interger < OFFFFFFFFH to an array of ASCII
   codes.
   Input variables are:  valure = integer to be converted,
                         base = number base to be used for conversion,
                         ld = leading character to be filled in,
                         bufadr = buffer address of the array,
                         width = size of array. */

declare value dword,
        bufadr pointer,
        (i, j, base, ld, width) byte,
        chars based bufadr (1) byte;

    do i = 1 to width;
        j = value mod base;
        if j < 10 then chars (width - i) = j + 30H;
        else chars (width - i) = j + 37H;
        value = value / base;
    end;
    i = 0;
    do while chars (i) = '0' and i < width - 1;
        chars (i) = ld;
        i = i + 1;
    end;
    char$count = width - i;

end int$to$asci;


out$word: procedure (w$ptr, distance);

/* An integer at (selector of w$ptr):(offset of w$ptr + distance) is printed
   as a 4 digit hexadecimal number.  */

declare chars(4) byte,
        w$ptr pointer,
        distance byte,
        w based w$ptr (1) word;

    call int$to$asci(w(distance), 16, '0', @chars(0), 4);
    call write(0, @chars(0), 4, @status);

end out$word;


write$int: procedure(dw, t);

/* An integer (dw) is printed in hexadecimal (t = 1) or in decimal (t = 0). */
```

292010-39

Traffic Simulator/Monitor Station Program (Continued)

```
declare dw dword,
        chars (10) byte,
        t byte;

    if t then
    do;
        call int$to$asci(dw, 16, 0, @chars(0), 8);
        call write(0, @chars(8-char$count), char$count, @status);
    end;
    else
    do;
        call int$to$asci(dw, 10, 0, @chars(0), 10);
        call write(0, @chars(10-char$count), char$count, @status);
    end;

end write$int;


out$dec$hex: procedure(dw);

/* This procedure prints an integer in decimal and hexadecimal. */

declare dw dword;

    call write$int(dw, 0);
    call write(0, @(' ('), 2, @status);
    call write$int(dw, 1);
    call write(0, @('H)'), 2, @status);

end out$dec$hex;


write$offset: procedure(w$ptr);

/* This procedure takes a pointer variable, converts it to a 82586 type offset,
   and prints it in hexadecimal.  */

declare w$ptr pointer,
        w word;

    call write(0, @(' at '), 4, @status);
    w = offset(w$ptr);
    call out$word(@w, 0);
    call write(0, @('  '), 2, @status);

end write$offset;


write$address: procedure (ptr);

/* This procedure takes a pointer variable and prints it in the
   'selector:offset' format.  */

declare (ptr, ptr$loc)  pointer,
        w based ptr$loc (2) word;

    ptr$loc = @ptr;
```

292010–40

Traffic Simulator/Monitor Station Program (Continued)

```
2           call out$word(@w(1), 0);
2           call write(0, @(':'), 1, @status);
2           call out$word(@w(0), 0);
2           call write(0, @(' '), 1, @status);

2       end write$address;


1       print$wds: procedure(w$ptr, no$words);

        /* This procedure prints no$words number of words starting at w$ptr.  */

2       declare w$ptr pointer,
                (i, no$words) byte;

2           if no$words <> 0 then
2           do;
3               call cr$lf;
3               do i = 0 to no$words - 1;
4                   call out$word(w$ptr, i);
4                   if i = 0 then
4                       call write$offset(w$ptr);
4                   call cr$lf;
4               end;
3           end;

2       end print$wds;


1       print$str: procedure (str$ptr, len);

        /* This procedure prints len number of bytes starting at str$ptr. */

2       declare (len, i) byte,
                chars (2) byte,
                str$ptr pointer,
                str based str$ptr (1) byte;

2           if len <> 0 then
2           do i = 0 to (len - 1);
3               call int$to$asci(str(i), 16, '0', @chars(0), 2);
3               call write(0, @chars(0), 2, @status);
3               call write(0, @('  '), 2, @status);
3           end;
2           call cr$lf;

2       end print$str;


1       print$buff: procedure (ptr, cnt);

        /* This procedure prints cnt number of buffer contents starting at ptr. */

2       declare ptr pointer,
                bt based ptr (1) byte,
                (i, j) byte,
                cnt word;
```

292010–41

Traffic Simulator/Monitor Station Program (Continued)

```
2          if cnt > 16 then
2          do;
3              i = shr(cnt, 4) - 1;
3              do j = 0 to i;
4                  call write$address(@bt(16*j));
4                  call print$str(@bt(16*j), 16);
4                  if (j = 20) or (j = 40) or (j = 60) or (j = 80) then
4                  call pause;
4              end;
3              i = i + 1;
3              if cnt-16*i <> 0 then call write$address(@bt(16*i));
3              call print$str(@bt(16*i), cnt-16*i);
3          end;
2          else
           do;
3              call write$address(@bt(0));
3              call print$str(@bt(0), cnt);
3          end;

2      end print$buff;



1      read$int: procedure (limit) dword;

       /* This procedure reads integer characters and forms an integer.  If the
          integer is bigger than 'limit' or an overflow error is encounterred, then
          an error message is printed.  */

2      declare (wd, wh, limit) dword,
               (i, j, k, done, hex, dover, hover) byte;

2          do forever;
3              call read(1, @c$buf, 80, @actual, @status);
3              i, k = skip;
3              hex, done, dover, hover = false;
3              wd, wh = 0;
3              j = char$to$int(c$buf(i));
3              do while j <= 15;
4                  if j > 9 then hex = true;
4                  if not dover then
4                      if wd > 429496729 then dover = true;
4                      else if (wd = 429496729) and (j > 5) then dover = true;
4                  wd = wd*10 + j;
4                  if not hover then if wh > 0FFFFFFFH then hover = true;
4                  wh = wh*16 + j;
4                  i = i + 1;
4                  j = char$to$int(c$buf(i));
4              end;
3              if ((c$buf(i) <> 'H') and (c$buf(i) <> 'h') and (c$buf(i) <> 0DH) and
                       (c$buf(i) <> 0AH) and (c$buf(i) <> ' ')) or (i = k) then
3              call writeln(0, @(0DH, 0AH, ' Illegal character'), 20, @status);
3              else
               do;
4                  if (c$buf(i) = 'H') or (c$buf(i) = 'h') then hex = true;
4                  if hex then
```

292010-42

Traffic Simulator/Monitor Station Program (Continued)

```
4                 do;
5                     if not hover and (wh <= limit) then return wh;
5                 end;
4                 else
4                     if not dover and (wd <= limit) then return wd;
4                 call writeln(0, @(0DH, 0AH, ' The number is too big.'), 25,
                                                                   @status);
4                 call write(0, @(' It has to be less than or equal to '), 36,
                                                                   @status);
4                 call out$dec$hex(limit);
4                 call writeln(0, @('.'), 1, @status);
4             end;
3             call write(0, @(' Enter a number ==> '), 20, @status);
3         end;

2     end read$int;


1     put$address: procedure(where);

      /* This procedure puts an address typed in hexadecimal to the specified
         location 'where'. */

2     declare where pointer,
              (i, j, m, err) byte,
              addr based where (1) byte;

2         do forever;
3             err = false;
3             call read(1, @c$buf, 80, @actual, @status);
3             i = skip;
3             m = address$length;
3             do while (m <> 0) and not err;
4                 j = char$to$int(c$buf(i));
4                 if j = 0FFH then err = true;
4                 else
                  do;
5                     addr(m-1) = shl(j, 4);
5                     j = char$to$int(c$buf(i+1));
5                     if j = 0FFH then err = true;
5                     else addr(m-1) = addr(m-1) or j;
5                 end;
4                 i = i + 2;
4                 m = m - 1;
4             end;
3             if not err then
3             do;
4                 m = c$buf(i);
4                 if (m = 0DH) or (m = 0AH) or (m = 'h') or (m = 'H') or (m = ' ')
4                     then return;
4             end;
3             call writeln(0, @(0DH, 0AH, ' Illegal character'), 20, @status);
3             call write(0, @(' Enter an address in Hex ==> '), 29, @status);
3         end;

2     end put$address;
```

292010–43

Traffic Simulator/Monitor Station Program (Continued)

```
percent: procedure;

/* This procedure calculates and prints a network percent load generated
   by this station.  The equation used in this procedure was obtained
   from actual measurements. */

declare i word,
        (j, k) dword,
        pcent (3) byte;

    j = (tbd.act$count and 3FFFH)*8;
    if not ad$loc then k = (2*address$length + 2 + crc + preamble)*8;
    else k = (crc + preamble)*8;
    if delay <> 0 then

$IF NOT SBC18651

    i = low((1000*(j + k))/(1805 + k + 5*double(delay) + j));

$ELSE

    i = low((1000*(j + k))/(2021 + k + 5*double(delay) + j));

$ENDIF

    else

$IF NOT SBC18651

    i = low((1000*(j + k))/(1810 + k + j));

$ELSE

    i = low((1000*(j + k))/(2026 + k + j));

$ENDIF

    call int$to$asci(i, 10, 0, @pcent(0), 3);
    call write(0, @pcent(0), 2, @status);
    call write(0, @('.'), 1, @status);
    call write(0, @pcent(2), 1, @status);
    call writeln(0, @(' %'), 2, @status);

end percent;


print$network$addr: procedure (ptr);

/* This station's address is printed with its least significant bit
   in the most right position.  */

declare ptr pointer,
        addr based ptr (1) byte,
        char (6) byte,
        i byte;
```

292010-44

Traffic Simulator/Monitor Station Program (Continued)

```
2             do i = 1 to address$length;
3                 char(i-1) = addr(address$length-i);
3             end;
2             call print$str(@char(0), address$length);

2         end print$network$addr;


1         print$parameters: procedure;

          /* This procedure prints transmission parameters. */

2         declare w dword,
                  stgs (6) byte;

2             call write(0, @(' Destination Address: '), 22, @status);
2             if not ad$loc then
2                 call print$network$addr(@transmit.dest$adr(0));
2             else
                  call print$network$addr(@tx$buffer(0));
2             if not ad$loc then
2             w =  (tbd.act$count and 3FFFH) + address$length * 2 + 2 + crc;
2             else w = (tbd.act$count and 3FFFH) + crc;
2             call write(0, @(' Frame Length: '), 15, @status);
2             call write$int(w, 0);
2             call writeln(0, @(' bytes'), 6, @status);
2             call write(0, @(' Time Interval between Transmit Frames: '), 40, @status);
2             if delay <> 0 then
2             do;
          $IF NOT SBC18651

3                 w = 1810 + (double(delay) - 1) * 5;

          $ELSE

                  w = 2026 + (double(delay) - 1) * 5;

          $ENDIF

3                 call int$to$asci(w, 10, 0, @stgs, 6);
3                 if w >= 10000 then
3                 do;
4                     call write(0, @stgs(0), 2, @status);
4                     call write(0, @('.'), 1, @status);
4                     call write(0, @stgs(2), 2, @status);
4                     call writeln(0, @(' miliseconds'), 12, @status);
4                 end;
3                 else
                  do;
4                     call write(0, @stgs(0), 5, @status);
4                     call write(0, @('.'), 1, @status);
4                     call write(0, @stgs(5), 1, @status);
4                     call writeln(0, @(' microseconds'), 13, @status);
4                 end;
3             end;
2             else
```

292010–45

Traffic Simulator/Monitor Station Program (Continued)

```
$IF NOT SBC18651

    call writeln(0, @(' 159.4 microseconds'), 19, @status);

$ELSE

    call writeln(0, @(' 172.8 microseconds'), 19, @status);

$ENDIF

    call write(0, @(' Network Percent Load generated by this station: '), 49,
                                                                  @status);
    call percent;
    call write(0, @(' Transmit Frame Terminal Count: '), 32, @status);
    if stop then call write$int(stop$count, dhex);
        else call write(0, @('Not Defined'), 11, @status);
    call cr$lf;

end print$parameters;


print$scb: procedure;

/* prints the SCB */

    call writeln(0, @(0DH, 0AH, '*** System Control Block ***'), 30, @status);
    call print$wds(@scb.status, 8);

end print$scb;


wait$scb: procedure;

/* This procedure provids a wait loop for the SCB command word to
   become cleared.  */

declare i word;

    i = 0;
    do while (scb.cmd <> 0) and (i < 8000H);
        i = i + 1;
    end;
    if scb.cmd <> 0 then
        do;
            call write(0, @(0DH, 0AH, ' Wait Time = '), 15, @status);
            call write$int(i, 0);
            call cr$lf;
        end;

end wait$scb;


start$timer0: procedure;

/* 80186 timer0 is started. */
```

292010–46

Traffic Simulator/Monitor Station Program (Continued)

```
2           output(TIMERO$CTL) = OE000H;

2       end start$timer0;


1       isr: procedure interrupt INT$TYPE$586 reentrant;

        /* interrupt service routine for 82586 interrupt */

2       declare i byte;

            /* Enable 82586 Interrupt */

        $IF SBC18651

            output (PIC$EOI$130) = EOI$CMDO$130;
            enable;

        $ELSE

2           output (PIC$EOI$186) = EOI$CMDO$186;
2           enable;

        $ENDIF

            /* Frame Received Interrupt has the highest priority */

2           if (scb.status and 4000H) = 4000H then
2           do;
3               disable;
3               scb.cmd = 4000H;
3               output (CA$PORT) = CA;
3               call wait$scb;
3               if rfd(current$frame).status = 0A000H then
3               do;
4                   receive$count = receive$count + 1;
4                   current$frame = current$frame + 1;
4                   if current$frame = 5 then current$frame = 0;
4               end;
3               return;
3           end;

2           if (scb.status and 2000H) = 2000H then
2           do;
3               disable;
3               scb.cmd = 2000H;
3               output(CA$PORT) = CA;
3               call wait$scb;
3               enable;
3               if (transmit.status and 0A000H) = 0A000H then
3               do;
4                   count = count + 1;
4                   if (stop and (count = stop$count)) then return;
4                   else
                    do;
```

292010–47

Traffic Simulator/Monitor Station Program (Continued)

```
                    transmit.status = 0;
                    if delay = 0 then
                    do;
                        disable;
                        scb.cmd = 0100H;
                        output(CA$PORT) = CA;
                        call wait$scb;
                        return;
                    end;
                    else
                    do;
                        call start$timer0;
                        return;
                    end;
                end;
            end;
            if (transmit.status and 0020H) = 0020H then
            do;
                transmit.status = 0;
                disable;
                scb.cmd = 0100H;
                output (CA$PORT) = CA;
                call wait$scb;
                return;
            end;
            if (transmit.status and 0400H) = 0400H then
            do;
                call write(0, @(0DH, ' No Carrier Sense!', 0DH), 20, @status);
                transmit.status = 0;
                disable;
                scb.cmd = 0100H;
                output (CA$PORT) = CA;
                call wait$scb;
                return;
            end;
            if (transmit.status and 0200H) = 0200H then
            do;
                call write(0, @(0DH, ' Lost Clear to Send!', 0DH), 22, @status);
                transmit.status = 0;
                disable;
                scb.cmd = 0100H;
                output (CA$PORT) = CA;
                call wait$scb;
                return;
            end;
            if (transmit.status and 0100H) = 0100H then
            do;
                call write(0, @(0DH, ' DMA Underrun!', 0DH), 16, @status);
                transmit.status = 0;
                disable;
                scb.cmd = 0100H;
                output (CA$PORT) = CA;
                call wait$scb;
                return;
            end;
        end;
        if (scb.status and 8000H) = 8000H then
```

292010–48

Traffic Simulator/Monitor Station Program (Continued)

```
    do;
        disable;
        scb.cmd = 8000H;
        output (CA$PORT) = CA;
        call wait$scb;
    end;
    if (scb.status and 1000H) = 1000H then
    do;
        disable;
        scb.cmd = 1000H;
        output (CA$PORT) = CA;
        call wait$scb;
        call write(0, @(0DH, ' Receive Unit became not ready.', 0DH), 33,
                                                                  @status);
    end;

    if reset then
    do;
        if iscp.busy then
        do;
            call writeln(0, @(0DH, 0AH, ' Reset failed.'), 16, @status);
            disable;
            scb.cmd = 0080H;
            output (CA$PORT) = CA;
            call wait$scb;
            output (CA$PORT) = CA;
            call writeln(0, @(' Software Reset Executed!'), 25, @status);
        end;
        else reset = false;
    end;

end isr;



tx$isr: procedure interrupt INT$TYPE$TIMER0;

/* interrupt survice routine for 80186 timer interrupt*/

    scb.cmd = 0100H;
    output(CA$PORT) = CA;
    call wait$scb;

$IF SBC18651

    output(PIC$EOI$130) = EOI$CMD4$130;
    enable;
    output(PIC$EOI$186) = EOI$CMD0$186;

$ELSE

    output(PIC$EOI$186) = EOI$CMD4$186;

$ENDIF

end tx$isr;
```

292010-49

Traffic Simulator/Monitor Station Program (Continued)

```
$IF SBC18651

isr$7: procedure interrupt INT$7;

/* The 80130 generates an interrupt 7 if the original interrupt is not
   active any more when the first interrupt acknowledge is received. */

    call write(0, @(0DH, 'Interrupt 7', 0DH), 13, @status);

end isr$7;

$ENDIF


read$byte: procedure (k) byte;
declare k word;

    call write(0, @(0DH, 0AH, ' Enter byte '), 14, @status);
    call out$dec$hex(k);
    call write(0, @(' ==> '), 5, @status);
    return read$int(0FFH);

end read$byte;


init$186$timer0: procedure;

/* This procedure initializes the 80186 timer 0.  */

declare i byte;

$IF SBC18651

    output(INT$CTL$TIMER0) = 8;
    call write(0, @(0DH, 0AH, ' Enter a delay count ==> '), 27, @status);
    delay = read$int(0FFFFH);
    if (delay < 100) and (delay <> 0) then
    do;
        call cr$lf;
        call cr$lf;
        call loop$char(35, '*');
        call write(0, @(' WARNING '), 9, @status);
        call loop$char(35, '*');
        call writeln(0, @(0DH, 0AH, 'A delay count between 0 and 100 may be very ',
                'dangerous when this station starts'), 80, @status);
        call writeln(0, @('to receive many frames separated only by the ',
                'IFS period (9.6 microseconds).'), 75, @status);
        call writeln(0, @('If this station never receives a frame, then ',
                     'ignore this warning.'), 65, @status);
        call loop$char(79, '*');
    end;
    output(MAX$COUNT$A) = delay;
    call cr$lf;
    output(PIC$MASK$186) = 3EH;
```

292010–50

Traffic Simulator/Monitor Station Program (Continued)

```
$ElSE

        output(INT$CTL$TIMERO) = OCH;
        call write(O, @(ODH, OAH, ' Enter a delay count ==> '), 27, @status);
        delay = read$int(OFFFFH);
        output(MAX$COUNT$A) = delay;
        call cr$lf;
        output(PIC$MASK$186) = ENABLE$586$186;

$ENDIF

end init$186$timerO;


setup$ia$parameters: procedure;
declare i byte;

        call write(O, @(ODH, OAH, ' Configure the 586 with the prewired'
                                       ,' board address ==> '), 57, @status);
        if yes then
        do i = O to address$length - 1;
            ia$setup.ia$address(i) = input(BOARD$ADDRESS$BASE + 10 - 2 * i);
        end;
        else
        do;
            call write(O, @(ODH, OAH, ' Enter this station''s address',
                                                 ' in Hex ==> '),43, @status);
            call put$address(@ia$setup.ia$address(O));
        end;

end setup$ia$parameters;


setup$mc$parameters: procedure;
declare (j, k, done) byte;

        j = O;
        call writeln(O, @(ODH, OAH, ' You can enter up to 8 Multicast Addresses.'),
                                                                    45, @status);
        done = false;
        call write(O, @(' Would you like to enter a Multicast Address?',
                                                ' (Y or N) ==> '), 59, @status);
        do while not done;
            if yes then
            do;
                k = j * address$length;
                j = j + 1;
                call cr$lf;
                if j = 9 then
                do;
                    call write(O, @(' You already entered 8 Multicast addresses.'),
                                                              43, @status);
                    done = true;
                end;
                else
                do;
                    call write(O, @(' Enter a Multicast Address ==> '), 31, @status);
```

292010-51

Traffic Simulator/Monitor Station Program (Continued)

```
5                       call put$address(@mc$setup.mc$address(k));
5                       call write(0, @(0DH, 0AH, ' More Multicast Addresses?',
                                                        ' (Y or N) ==> '), 42, @status);
5                   end;
4               end;
3               else done = true;
3           end;
2           if j = 9 then j = j - 1;
2           mc$count = address$length * j;
2           mc$setup.mc$byte$count = mc$count;
2           call write(0, @(0DH, 0AH, ' You entered '), 15, @status);
2           call write$int(j, 0);
2           call writeln(0, @(' Multicast Address(es). '), 23, @status);

2       end setup$mc$parameters;


1       setup$configure$parameters: procedure;
2       declare (k, j) byte;

2           configure.byte$cnt = 11;
2           configure.info(0) = 8;
2           configure.info(1) = 0;
2           configure.info(2) = 26H;
2           configure.info(3) = 0;
2           configure.info(4) = 96;
2           configure.info(5) = 0;
2           configure.info(6) = 0F2H;
2           configure.info(7) = 0;
2           configure.info(8) = 0;
2           configure.info(9) = 64;
2           j = 0;
2           call write(0, @(0DH, 0AH, ' Configure command is set up for default',
                ' values. ', 0DH, 0AH, ' Do you want to change any bytes?',
                                                        ' (Y or N) ==> '), 99, @status);
2           do while yes;
3               do while j = 0;
4                   call write(0, @(0DH, 0AH, ' Enter byte number (1 - 11) ==> '), 34,
                                                                        @status);
4                   j = read$int(11);
4                   if j = 0 then
4                   call write(0, @(0DH, 0AH, ' Illegal byte number'), 22, @status);
4               end;
3               if j = 1 then configure.byte$cnt = read$byte(j);
3                   else configure.info(j - 2) = read$byte(j);
3               j = 0;
3               call write(0, @(0DH, 0AH, ' Any more bytes? (Y or N) ==> '), 32,
                                                                        @status);
3           end;
2           preamble = shl(1, shr((configure.info (2) and 30H), 4)+1);
2           address$length = configure.info(2) and 07H;
2           if address$length = 7 then address$length = 0;
2           ad$loc = shr((configure.info(2) and 08H), 3);
2           if shr((configure.info(7) and 20H), 5) then crc = 2; else crc = 4;
2           if shr((configure.info(7) and 10H), 4) then crc = 0;

2       end setup$configure$parameters;
```

292010-52

Traffic Simulator/Monitor Station Program (Continued)

```
1       setup$tx$parameters: procedure;
2       declare (size, i) word;

2           do forever;
3               no$transmission = false;
3               transmit.bd$offset = offset (@tbd.act$count);
3               if not ad$loc then
3               do;
4                   call write(0, @(0DH, 0AH,
                        ' Enter a destination address in Hex ==> '), 42, @status);
4                   call put$address(@transmit.dest$adr(0));
4               end;
3               else call writeln(0, @(' 82586 is configured to pick up DA, IA, ',
                                  ' and TYPE from TX buffer. '), 64, @status);
3               call cr$lf;
3               if not ad$loc then
3               do;
4                   call write(0, @(0DH, 0AH, ' Enter TYPE ==> '), 18, @status);
4                   transmit.type = read$int(0FFFFH);
4               end;
3               call writeln(0, @(0DH, 0AH,' How many bytes of transmit data?'), 35,
                                                                          @status);
3               call write(0, @(' Enter a number ==> '), 20, @status);
3               size = read$int(1518);
3               tbd.act$count = size or 8000H;
3               if size <> 0 then
3               do;
4                   tbd.link$offset = 0FFFFH;
4                   tbd.ad0 = offset (@tx$buffer(0));
4                   tbd.ad1 = 0;
4                   do i = 0 to 1517;
5                       tx$buffer(i) = i;
5                   end;
4                   call writeln(0,
                    @(0DH, 0AH, ' Transmit Data is continuous numbers (0, 1, 2, 3, ',
                                                       ' ... )'), 57, @status);
4                   call write(0, @(' Change any data bytes? (Y or N) ==> '), 37,
                                                                          @status);
4                   do while yes;
5                       call write(0, @(0DH, 0AH, ' Enter a byte number ==> '),
                                                                    27, @status);
5                       i = read$int(size);
5                       call write(0, @(0DH, 0AH, ' Byte '), 8, @status);
5                       call out$dec$hex(i);
5                       call write(0, @(' currently contains '), 20, @status);
5                       call out$dec$hex(tx$buffer(i));
5                       call write(0, @('.'), 1, @status);
5                       tx$buffer(i) = read$byte(i);
5                       call write(0, @(0DH, 0AH, ' Any more bytes? (Y or N) ==> '),
                                                                    32, @status);
5                   end;
4               end;
3               else
                transmit.bd$offset = 0FFFFH;
3               call cr$lf;
```

292010-53

Traffic Simulator/Monitor Station Program (Continued)

```
3            call init$186$timer0;
3            call write(0, @(0DH, 0AH, ' Setup a transmit terminal count?',
                                       ' (Y or N) ==> '), 49, @status);
3            if yes then
3            do;
4                stop = true;
4                call write (0, @(0DH, 0AH, ' Enter a transmit',
                                       ' terminal count ==> '), 39, @status);
4                stop$count = read$int(0FFFFFFFFH);
4            end;
3            else stop = false;
3            call cr$lf;
3            call cr$lf;
3            call print$parameters;
3            call write(0, @(0DH, 0AH, ' Good enough? (Y or N) ==> '), 29,
                                                                  @status);
3            if yes then return;
3        end;

2    end setup$tx$parameters;


1    loop$char: procedure (i, j);
2    declare (i, j, k) byte;

2        do k = 1 to i;
3            call write(0, @j, 1, @status);
3        end;

2    end loop$char;


1    init: procedure;

2    declare  i byte;

2        call cr$lf;
2        call loop$char(13, 0AH);
2        call loop$char(15, ' ');
2        call writeln(0, @('TRAFFIC SIMULATOR AND MONITOR',
                                 ' STATION PROGRAM '), 46, @status);
2        call loop$char(7, 0AH);
2        call writeln(0, @(0DH, 0AH, ' Initialization begun'), 23, @status);
2        call cr$lf;
2        reset = true;
2        cur$cb$offset = 0FFFFH;
2        output(ESI$PORT) = NO$LOOPBACK;
2        output(ESI$PORT) = LOOPBACK;
2        dhex = false;

         /* set up interrupt logic */

2        call set$interrupt(INT$TYPE$586, isr);
2        call set$interrupt(INT$TYPE$TIMER0, tx$isr);

     $IF SBC18651
```

292010-54

Traffic Simulator/Monitor Station Program (Continued)

```
                 call set$interrupt(INT$7, isr7);
                 output (PIC$MASK$130) = ENABLE$586$186;
                 output (PIC$EOI$130) = EOI$CMDO$130;
                 output (PIC$EOI$130) = EOI$CMD4$130;
                 output (PIC$EOI$186) = EOI$CMDO$186;
                 output (PIC$VTR$186) = 30H;

             $ELSE

     2           output (PIC$EOI$186) = EOI$CMDO$186;
     2           output (PIC$EOI$186) = EOI$CMD4$186;
     2           output (PIC$MASK$186) = ENABLE$586;

             $ENDIF

                 /* locate iscp */

     2           iscp$ptr = ISCP$LOC$LO;

                 /* set up fields in ISCP */

     2           iscp.busy = 1;
     2           iscp.scb$b(0) = SCB$BASE$LO;
     2           iscp.scb$b(1) = SCB$BASE$HI;
     2           iscp.scb$o = offset (@scb.status);

                 /* set up SCB */

     2           scb.status = 0;
     2           scb.cbl$offset = offset (@diagnose.status);
     2           scb.rpa$offset = offset (@rfd(0).status);
     2           scb.crc$errs = 0;
     2           scb.aln$errs = 0;
     2           scb.rsc$errs = 0;
     2           scb.ovrn$errs = 0;

                 /* set up Diagnose command */

     2           diagnose.status = 0;
     2           diagnose.cmd = 7;
     2           diagnose.link$offset = offset (@configure.status);

                 /* set up CONFIGURE command */

     2           configure.status = 0;
     2           configure.cmd = 2;
     2           configure.link$offset = offset (@ia$setup.status);;
     2           call setup$configure$parameters;


                 /* set up IA command */

     2           ia$setup.status = 0;
     2           ia$setup.cmd = 1;
     2           ia$setup.link$offset = offset (@mc$setup.status);
     2           call setup$ia$parameters;
```

292010-55

Traffic Simulator/Monitor Station Program (Continued)

```
/* set up MC command */

mc$setup.status = 0;
mc$setup.cmd = 8003H;
mc$setup.link$offset = 0FFFFH;
call setup$mc$parameters;


/* set up one transmit cb linked to itself */

transmit.status = 0;
call writeln(0, @(0DH, 0AH, ' Would you like to transmit?'), 30, @status);
call write(0, @(' Enter a Y or N ==> '), 20, @status);
if yes then
do;
    transmit.cmd = 8004H;
    transmit.link$offset = 0FFFFH;
    transmit.bd$offset = offset (@tbd.act$count);
    call setup$tx$parameters;
end;
else no$transmission = true;


/* initialize receive packet area */

do i = 0 to 3;
    rfd(i).status = 0;
    rfd(i).el$s = 0;
    rfd(i).link$offset = offset (@rfd(i+1).status);
    rfd(i).bd$offset = 0FFFFH;
    rbd(i).act$count = 0;
    rbd(i).next$bd$link = offset (@rbd(i+1).act$count);
    rbd(i).ad0 = offset (@rbuf(i).buffer(0));
    rbd(i).ad1 = 0;
    rbd(i).size = 1500;
end;
rfd(0).bd$offset = offset (@rbd(0).act$count);
rfd(4).status = 0;
rfd(4).el$s = 0;
rfd(4).link$offset = offset (@rfd(0).status);
rfd(4).bd$offset = 0FFFFH;
rbd(4).act$count = 0;
rbd(4).next$bd$link = offset (@rbd(0).act$count);
rbd(4).ad0 = offset (@rbuf(4).buffer(0));
rbd(4).ad1 = 0;
rbd(4).size = 1500;


/* initialize counters */

count = 0;
receive$count = 0;
current$frame = 0;

/* issue the first CA */
```

292010-56

Traffic Simulator/Monitor Station Program (Continued)

```
        output(CA$PORT) = CA;

    end init;


    print$help: procedure;

        call writeln(0, @(0DH, 0AH, ' Commands are: '), 16, @status);
        call writeln(0, @(0DH, 0AH, ' S - Setup CB              D - Display RFD/CB'),
                                                               45, @status);
        call writeln(0, @(' P - Print SCB           C - SCB Control CMD'), 44,
                                                                   @status);
        call writeln(0, @(' L - ESI Loopback On     N - ESI Loopback Off'), 45,
                                                                   @status);
        call writeln(0, @(' A - Toggle Number Base'), 23,
                                                                   @status);
        call writeln(0, @(' Z - Clear Tx Frame Counter'), 27, @status);
        call writeln(0, @(' Y - Clear Rx Frame Counter'), 27, @status);
        call writeln(0, @(' E - Exit to Continuous Mode'), 28, @status);

    end print$help;



    enter$scb$cmd: procedure;
    declare i byte;

    /* enter a command into the SCB */

        call cr$lf;
        if scb.cmd <> 0 then
        do;
            call writeln(0, @(' SCB command word is not cleared'), 32, @status);
            call write(0, @(' Try a Channel Attention? (Y or N) ==> '),
                                                               39, @status);
            if yes then
            do;
                output(CA$PORT) = CA;
                call writeln(0, @(' Issued channel attention'), 25, @status);
                call cr$lf;
                return;
            end;
        end;
        call write(0, @(' Do you want to enter any SCB commands? (Y or N) ==> '),
                                                        53, @status);
        if not yes then return;
        call write(0, @(0DH, 0AH, ' Enter CUC ==> '), 17, @status);
        i = read$int(4);
        scb.cmd = scb.cmd or shl(double(i), 8);
        if i = 1 then scb.cbl$offset = cur$cb$offset;
        call write(0, @(0DH, 0AH, ' Enter RES bit ==> '), 21, @status);
        i = read$bit;
        scb.cmd = scb.cmd or shl(i, 7);
        call write(0, @(0DH, 0AH, ' Enter RUC ==> '), 17, @status);
        i = read$int(4);
        scb.cmd = scb.cmd or shl(i, 4);
```

292010-57

Traffic Simulator/Monitor Station Program (Continued)

```
        if (((scb.cbl$offset = offset (@transmit.status))
            and ((scb.cmd and 0100H) = 0100H)) or ((scb.cmd and 0010H) = 0010H))
            and not ((scb.cmd and 0080H) = 0080H)
            then goback = 1;
        call writeln(0, @(0DH, 0AH, ' Issued Channel Attention'), 27, @status);
        call cr$lf;
        output(CA$PORT) = CA;

    end enter$scb$cmd;


    print$type$help: procedure;

        call writeln(0, @(0DH, 0AH, 0AH, 'Command block type: '), 22, @status);
        call writeln(0, @(' N - Nop                 I - IA Setup'), 35, @status);
        call writeln(0, @(' C - Configure           M - MA Setup'), 35, @status);
        call writeln(0, @(' T - Transmit            R - TDR'), 30, @status);
        call writeln(0, @(' D - Diagnose            S - Dump Status'), 38, @status);
        call writeln(0, @(' H - Print this message'), 23, @status);

    end print$type$help;


    setup$cb: procedure;
    declare (t, valid) byte;

        valid = false;
        do while not valid;
            call write(0, @(0DH, 0AH, ' Enter command block type (H for',
                                            ' help) ==> '), 45, @status);
            t = read$char;
            if (t <> 'H') and (t <> 'h') and (t <> 'T') and (t <> 't') and
               (t <> 'N') and (t <> 'n') and (t <> 'R') and (t <> 'r') and
               (t <> 'D') and (t <> 'd') and (t <> 'C') and (t <> 'c') and
               (t <> 'I') and (t <> 'i') and (t <> 'M') and (t <> 'm') and
               (t <> 'S') and (t <> 's') then
            call write(0, @(0DH, 0AH, ' Illegal command block type'), 29,
                                                                  @status);
            else
            if (t = 'H') or (t = 'h') then call print$type$help;
            else valid = true;
        end;
        if (t = 'N') or (t = 'n') then
        do;
            cur$cb$offset = offset (@nop.status);
            nop.status = 0;
            nop.cmd = 8000H;
            nop.link$offset = 0FFFFH;
        end;
        if (t = 'I') or (t = 'i') then
        do;
            cur$cb$offset = offset (@ia$setup.status);
            ia$setup.status = 0;
            ia$setup.cmd = 8001H;
            ia$setup.link$offset = 0FFFFH;
            call setup$ia$parameters;
        end;
```

292010-58

Traffic Simulator/Monitor Station Program (Continued)

```
    if (t = 'C') or (t = 'c') then
    do;
        cur$cb$offset = offset (@configure.status);
        configure.status = 0;
        configure.cmd = 8002H;
        configure.link$offset = 0FFFFH;
        call setup$configure$parameters;
    end;
    if (t = 'M') or (t = 'm') then
    do;
        cur$cb$offset = offset (@mc$setup.status);
        mc$setup.status = 0;
        mc$setup.cmd = 8003H;
        mc$setup.link$offset = 0FFFFH;
        call setup$mc$parameters;
    end;
    if (t = 'T') or (t = 't') then
    do;
        cur$cb$offset = offset (@transmit.status);
        transmit.status = 0;
        transmit.cmd = 8004H;
        transmit.link$offset = 0FFFFH;
        call setup$tx$parameters;
    end;
    if (t = 'R') or (t = 'r') then
    do;
        cur$cb$offset = offset (@tdr.status);
        tdr.status = 0;
        tdr.cmd = 8005H;
        tdr.link$offset = 0FFFFH;
        tdr.result = 0;
    end;
    if (t = 'S') or (t = 's') then
    do;
        cur$cb$offset = offset (@dump.status);
        dump.status = 0;
        dump.cmd = 8006H;
        dump.link$offset = 0FFFFH;
        dump.buff$ptr = offset (@dump$area(0));
    end;
    if (t = 'D') or (t = 'd') then
    do;
        cur$cb$offset = offset (@diagnose.status);
        diagnose.status = 0;
        diagnose.cmd = 8007H;
        diagnose.link$offset = 0FFFFH;
    end;

end setup$cb;


display$command$block: procedure;
declare (i, j) byte,
        wh pointer,
        sel selector,
        w word;
```

292010-59

Traffic Simulator/Monitor Station Program (Continued)

```
        call cr$lf;
        if cur$cb$offset = OFFFFH then
            call write(0, @(' No Command Block to display'), 28, @status);
        if cur$cb$offset = offset (@nop.status) then
        do;
            call write(0, @('---NOP Command Block---'), 23, @status);
            call print$wds(@nop.status, 3);
        end;
        if cur$cb$offset = offset (@tdr.status) then
        do;
            call write(0, @('---TDR Command Block---'), 23, @status);
            call print$wds(@tdr.status, 4);
        end;
        if cur$cb$offset = offset (@diagnose.status) then
        do;
            call write(0, @('---Diagnose Command Block---'), 28, @status);
            call print$wds(@diagnose.status, 3);
        end;
        if cur$cb$offset = offset (@transmit.status) then
        do;
            call write(0, @('---Transmit Command Block---'), 28, @status);
            if not address$length then i = address$length;
            else i = address$length + 1;
            if ad$loc then call print$wds(@transmit.status, 4);
            else call print$wds(@transmit.status, i/2+1);
            call cr$lf;
            call cr$lf;
            if transmit.bd$offset <> OFFFFH then
            do;
                call write(0, @('---Transmit Buffer Descriptor---'), 33, @status);
                call print$wds(@tbd.act$count, 4);
                call write(0, @(0DH, 0AH, 0AH,
                    ' Display the transmit buffer? (Y or N) ==> '), 46, @status);
                if yes then
                do;
                    call cr$lf;
                    call writeln(0, @(' Transmit Buffer: '), 17, @status);
                    w = tbd.act$count and 3FFFH;
                    call print$buff(@tx$buffer(0), w);
                end;
            end;
        end;
        if cur$cb$offset = offset (@ia$setup.status) then
        do;
            call write(0, @('---IA Setup Command Block---'), 28, @status);
            call print$wds(@ia$setup.status, 6);
        end;
        if cur$cb$offset = offset (@configure.status) then
        do;
            call write(0, @('---Configure Command Block---'), 29, @status);
            call print$wds(@configure.status, 9);
        end;
        if cur$cb$offset = offset (@mc$setup.status) then
        do;
            call write(0, @('---MC Setup Command Block---'), 28, @status);
            i = 4 + mc$count/2;
```

292010-60

Traffic Simulator/Monitor Station Program (Continued)

```
                if mc$count > 24 then
                    do;
                        call print$wds(@mc$setup.status, 16);
                        call pause;
                        i = i - 16;
                        call print$wds(@mc$setup.mc$address(8), i);
                    end;
                else call print$wds(@mc$setup.status, i);
            end;
            if cur$cb$offset = offset (@dump.status) then
            do;
                call write(0, @('---Dump Status Command Block---'), 31, @status);
                call print$wds(@dump.status, 4);
                if dump.status = 0A000H then
                do;
                    call writeln(0, @(0DH, 0AH, ' Dump Status Results'), 22, @status)
                    call write$offset(@dump$area(0)); '
                    call cr$lf;
                    do i = 0 to 9;
                        call print$str(@dump$area(16*i), 16);
                    end;
                    call print$str(@dump$area(160), 10);
                    call cr$lf;
                end;
            end;

        end display$command$block;


        display$receive$area: procedure;
        declare (i, k, j, l) byte,
                chars(4) byte;

            call writeln(0, @(0DH, 0AH, ' Frame Descriptors: '), 21, @status);
            if ad$loc then
            do;
                call writeln(0, @(0DH, 0AH, ' DA, SA, and TYPE are in buffer.', 0DH,
                                                                        0AH), 36, @status);
                j = 3;
            end;
            else j = address$length + 4;
            do k = 0 to j;
                do i = 0 to 4;
                    call out$word(@rfd(i).status, k);
                    if k = 0 then call write$offset(@rfd(i).status);
                        else call loop$char(10, ' ');
                end;
                call cr$lf;
            end;
            call writeln(0, @(0DH, 0AH, 0AH, ' Receive Buffer Descriptors: '), 31,
                                                                              @status);
            do k = 0 to 4;
                do i = 0 to 4;
                    call out$word(@rbd(i).act$count, k);
                    if k = 0 then call write$offset(@rbd(i).act$count);
                        else call loop$char(10, ' ');
                end;
```

292010–61

Traffic Simulator/Monitor Station Program (Continued)

```
 3               call cr$lf;
 3           end;
 2           call write(0, @(0DH, 0AH, 0AH, ' Display the receive',
                                               ' buffers? (Y or N) ==> '), 46, @status);
 2           if not yes then return;
 2           call writeln(0, @(0DH, 0AH, ' Receive Buffers.'), 19, @status);
 2           do i = 0 to 4;
 3               call write(0, @(0DH, 0AH, ' Receive Buffer '), 18, @status);
 3               call write$int(i, 0);
 3               call writeln(0, @(' : '), 2, @status);
 3               k = rbd(i).act$count and 3FFFH;
 3               call print$buff(@rbuf(i).buffer(0), k);
 3               call pause;
 3           end;

 2       end display$receive$area;


 1       display$cb$rpa: procedure;
 2       declare i byte;

 2           call write(0, @(0DH, 0AH, ' Command Block or Receive Area? (R or C) ==> '),
                                                                            47, @status);
 2           i = read$char;
 2           do while (i <> 'R') and (i <> 'r') and (i <> 'C') and (i <> 'c');
 3               call writeln(0, @(0DH, 0AH, ' Illegal command'), 18, @status);
 3               call write(0, @(' Enter R or C ==> '), 18, @status);
 3               i = read$char;
 3           end;
 2           if (i = 'R') or (i = 'r') then call display$receive$area;
 2           else call display$command$block;

 2       end display$cb$rpa;


 1       process$cmd: procedure;
 2       declare (b, i) byte;

 2           goback = 0;
 2           b = read$char;
 2           call cr$lf;
 2           if (b <> 'H') and (b <> 'h') and (b <> 'S') and (b <> 's') and
                (b <> 'D') and (b <> 'd') and (b <> 'P') and (b <> 'p') and
                (b <> 'C') and (b <> 'c') and (b <> 'E') and (b <> 'e') and
                (b <> 'L') and (b <> 'l') and (b <> 'N') and (b <> 'n') and
                (b <> 'Z') and (b <> 'z') and (b <> 'Y') and (b <> 'y') and
                (b <> 'A') and (b <> 'a') then
 2           call write(0, @(' Illegal command'), 16, @status);
 2           if (b = 'H') or (b = 'h') then call print$help;
 2           if (b = 'A') or (b = 'a') then
 2               if dhex then
 2               do;
 3                   dhex = false;
 3                   call write(0, @(' Counters are displayed in decimal.'), 35,
                                                                              @status);
 3               end;
 2               else
```

292010-62

Traffic Simulator/Monitor Station Program (Continued)

```
                do;
                    dhex = true;
                    call write(0, @(' Counters are displayed in hexadecimal. '), 39,
                                                                        @status);
                end;
            if (b = 'L') or (b = 'l') then
            do;
                output(ESI$PORT) = LOOPBACK;
                call write(0, @(' ESI is in Loopback Mode. '), 25, @status);
            end;
            if (b = 'N') or (b = 'n') then
            do;
                output(ESI$PORT) = NO$LOOPBACK;
                call write(0, @(' ESI is NOT in Loopback Mode. '), 29, @status);
            end;
            if (b = 'Z') or (b = 'z') then
            do;
                count = 0;
                call write(0, @(' Transmit Frame Counter is cleared. '), 35, @status);
            end;
            if (b = 'Y') or (b = 'y') then
            do;
                receive$count = 0;
                scb.crc$errs, scb.aln$errs, scb.rsc$errs, scb.ovrn$errs = 0;
                call write(0, @(' Receive Frame Counter is cleared. '), 34, @status);
            end;
            if (b = 'C') or (b = 'c') then call enter$scb$cmd;
            if (b = 'S') or (b = 's') then call setup$cb;
            if (b = 'P') or (b = 'p') then call print$scb;
            if (b = 'D') or (b = 'd') then call display$cb$rpa;
            if (b = 'E') or (b = 'e') then goback = 1;
            call cr$lf;

        end process$cmd;


        getout: procedure;
        declare b byte;

            b = read$char;
            goback = 0;
            call write(0, @(0DH, 0AH, ' Enter command (H for help) ==> '), 34,
                                                                        @status);
            do forever;
                if csts then
                do;
                    disable;
                    call process$cmd;
                    enable;
                    if goback then return;
                    call write(0, @(0DH, 0AH, ' Enter command (H for help) ==> '), 34,
                                                                        @status);
                end;
            end;

        end getout;
```

292010–63

Traffic Simulator/Monitor Station Program (Continued)

```
1       update: procedure;
2       declare i byte;

2           call cr$lf;
2           call loop$char(10, 0AH);
2           call loop$char(28, '*');
2           call write(0, @(' Station Configuration '), 23, @status);
2           call loop$char(27, '*');
2           call cr$lf;
2           call cr$lf;
2           call write(0, @(' Host Address: '), 15, @status);
2           call print$network$addr(@ia$setup.ia$address(0));
2           i = 0;
2           call write(0, @(' Multicast Address(es): '), 24, @status);
2           if mc$setup.mc$byte$count = 0
2           then call writeln(0, @('No Multicast Addresses Defined'), 30, @status);
2           else
                do while i < mc$setup.mc$byte$count;
3                   call print$network$addr(@mc$setup.mc$address(i));
3                   call loop$char(24, ' ');
3                   i = i + 6;
3               end;
2           call write(0, @(0DH), 1, @status);
2           if not no$transmission then call print$parameters;
2           call write(0, @(' 82586 Configuration Block: '), 28, @status);
2           call print$str(@configure.info(0), 10);
2           call cr$lf;
2           call loop$char(29, '*');
2           call write(0, @(' Station Activities '), 20, @status);
2           call loop$char(29, '*');
2           call cr$lf;
2           call cr$lf;
2           call writeln(0,
        @(' # of Good     # of Good     CRC           Alignment     No            Receive'),
                                                                              73, @status);
2           call writeln(0,
        @(' Frames        Frames        Errors        Errors        Resource      Overrun'),
                                                                              73, @status);
2           call writeln(0,
        @(' Transmitted   Received                                  Errors        Errors'),
                                                                              72, @status);

2       end update;


1       main:

            call init;
1           enable;
1           do while reset;
2           end;
1           disable;
1           scb.cmd = 0100H;
1           output(CA$PORT) = CA;
1           call wait$scb;
1           enable;
```

292010-64

Traffic Simulator/Monitor Station Program (Continued)

```
 1          do while (diagnose.status and 8000H) <> 8000H;
 2          end;
 1          call cr$lf;
 1          if diagnose.status <> 0A000H
 1          then call writeln(0, @(' Diagnose failed!'), 17 , @status);
 1          if configure.status <> 0A000H
 1          then call writeln(0, @(' Configure failed!'), 18 , @status);
 1          if ia$setup.status <> 0A000H
 1          then call writeln(0, @(' IA Setup failed!'), 17 , @status);
 1          if mc$setup.status <> 0A000H
 1          then call writeln(0, @(' MC Setup failed!'), 17 , @status);
 1          scb.cbl$offset = offset (@transmit.status);
 1          call writeln(0, @(0DH, 0AH, ' Receive Unit is active.'), 26, @status);
 1          disable;
 1          scb.cmd = 0010H;
 1          output(CA$PORT) = CA;
 1          call wait$scb;
 1          enable;
 1          output(ESI$PORT) = NO$LOOPBACK;
 1          call cr$lf;
 1          if not no$transmission then
 1          do;
 2              call write(0, @('---Transmit Command Block---'), 28, @status);
 2              call print$wds(@transmit.status, 8);
 2              call cr$lf;
 2              cur$cb$offset = offset (@transmit.status);
 2              call pause;
 2              do z = 1 to 60;
 3                  call time(250);
 3              end;
 2              call writeln(0, @(0DH, 0AH, 'transmission started!'), 23, @status);
 2              call cr$lf;
 2              disable;
 2              scb.cmd = 0100H;
 2              output (CA$PORT) = CA;
 2              call wait$scb;
 2              enable;
 2          end;
 1          call update;
 1          do forever;
 2              call write(0, @(0DH, ' '), 2, @status);
 2              do y = 0 to 5;
 3                  do case y;
 4                      call write$int(count, dhex);
 4                      call write$int(receive$count, dhex);
 4                      call write$int(scb.crc$errs, dhex);
 4                      call write$int(scb.aln$errs, dhex);
 4                      call write$int(scb.rsc$errs, dhex);
 4                      call write$int(scb.ovrn$errs, dhex);
 4                  end;
 3                  char$count = 13 - char$count;
 3                  call loop$char(char$count, ' ');
 3              end;
 2              if csts then
 2              do;
 3                  disable;
 3                  call getout;
 3                  call update;
 3              end;
 2          end;

 1      end tsms;
```

292010-65

```
MODULE INFORMATION:

     CODE AREA SIZE     = 23C3H   9155D
     CONSTANT AREA SIZE = 0F85H   3973D
     VARIABLE AREA SIZE = 265EH   9822D
     MAXIMUM STACK SIZE = 0092H    146D
     1994 LINES READ
     0 PROGRAM WARNINGS
     0 PROGRAM ERRORS

DICTIONARY SUMMARY:

     159KB MEMORY AVAILABLE
     23KB MEMORY USED   (14%)
     0KB DISK SPACE USED

END OF PL/M-86 COMPILATION
```

292010-66

Traffic Simulator/Monitor Station Program (Continued)

```
        /*************************************************************************/
        /*                                                                       */
        /*          186/586 High Integration Board Initialization Routine        */
        /*          (This driver is configured for Ethernet/Cheapernet Design    */
        /*           Kit Demo Board)                                             */
        /*                                                                       */
        /*          Ver. 2.0                          March 14, 1986             */
        /*                                                                       */
        /*          Kiyoshi Nishide                   Intel Corporation          */
        /*                                                                       */
        /*************************************************************************/

        /* The conditional compilation parameter 'EPROM27128' determines board ROM
           size.  If it is true, the 80186's wait state generator is programmed to
           0 wait state for upper 64K-byte memory locations.  If it is false, the
           wait state generator is programmed to 0 wait state for upper 128K-byte
           memory locations   */


1       ini186:

        do;

2   1   declare hib_ir label public;
3   1   declare main label external;
4   1   declare menu label external;

        /* literals */

5   1   declare lit       literally 'literally',
                UMCS_reg         lit 'OFFA0H',
                LMCS_reg         lit 'OFFA2H',
                PACS_reg         lit 'OFFA4H',
                MPCS_reg         lit 'OFFA8H',
                INT_MASK_reg     lit 'OFF28H',
                ISCP$LOC$LO      lit '03FF8H',
                ISCP$LOC$HI      lit '0',
                SCC_CH_B_CMD     lit '8300H',
                SCC_CH_B_DATA    lit '8302H',
                SCC_CH_A_CMD     lit '8304H',
                SCC_CH_A_DATA    lit '8306H',
                NUL              lit '0',
                CR               lit '0DH',
                LF               lit '0AH',
                BS               lit '08H',
                SP               lit '20H',
                QM               lit '3FH',
                DEL              lit '07FH',
                BEL              lit '07H';
```

292010-67

186/586 High Integration Board Initialization Routine

```
/* System Configuration Pointer */

declare scp structure
        (
        sysbus byte,
        unused (5) byte,
        iscp$addr$lo word,
        iscp$addr$hi word
        )
        at (OFFFF6H) data (0, 0, 0, 0, 0, 0, ISCP$LOC$LO, ISCP$LOC$HI);

init$int$clt: procedure;

        output(INT_mask_reg) = OFFH; /* mask all interrupts */

end init$int$clt;

rra: procedure (reg_no) byte;
declare reg_no byte;

    if (reg_no and OFH) <> 0 then output(SCC_CH_A_CMD) = reg_no and OFH;
    return input(SCC_CH_A_CMD);

end rra;

rrb: procedure (reg_no) byte;
declare reg_no byte;

    if (reg_no and OFH) <>0 then output (SCC_CH_B_CMD) = reg_no and OFH;
    return input(SCC_CH_B_CMD);

end rrb;

wra: procedure (reg_no, value);
declare (reg_no, value) byte;

    if (reg_no and OFH) <> 0 then output (SCC_CH_A_CMD) = reg_no and OFH;
    output (SCC_CH_A_CMD) = value;

end wra;

wrb: procedure (reg_no, value);
declare (reg_no, value) byte;

    if (reg_no and OFH) <> 0 then output (SCC_CH_B_CMD) = reg_no and OFH;
    output (SCC_CH_B_CMD) = value;

end wrb;

init$SCC$B: procedure;

    call wrb(09, 01000000b);   /* channel B reset */
```

292010-68

186/586 High Integration Board Initialization Routine (Continued)

```
2          call wrb(04, 01001110b);   /* 2 stop, no parity, brf = 16x */
2          call wrb(03, 11000000b);   /* rx 8 bits/char, no auto-enable */
2          call wrb(05, 01100000b);   /* tx 8 bits/char */
2          call wrb(10, 00000000b);
2          call wrb(11, 01010110b);   /* rxc = txc = BRG, trxc = BRG out */
2          call wrb(12, 00001011b);   /* baud rate = 9600 */
2          call wrb(13, 00000000b);
2          call wrb(14, 00000011b);   /* BRG source = SYS CLK, enable BRG */
2          call wrb(15, 00000000b);   /* all ext status interrupts off */

2          call wrb(03, 11000001b);   /* scc-b receive enable */
2          call wrb(05, 11101010b);   /* scc-b transmit enable, dtr on, rts on */

2      end init$SCC$B;


1      c$in: procedure byte public;

3          do while (input(SCC_CH_B_CMD) and 1) = 0; end;
2          return (input(SCC_CH_B_DATA));

2      end c$in;


1      c$out: procedure (char) public;
2      declare char byte;

3          do while (input(SCC_CH_B_CMD) and 4) = 0; end;
2          output(SCC_CH_B_DATA) = char;

2      end c$out;



1      read: procedure (file$id, msg$ptr, count, actual$ptr, status$ptr) public;
2      declare file$id word,
               msg$ptr pointer,
               count word,
               actual$ptr pointer,
               status$ptr pointer,
               msg based msg$ptr (1) byte,
               buf (200) byte,
               actual based actual$ptr word,
               status based status$ptr word,
               i word,
               ch byte;

       /* This procedure implements the ISIS read procedure. All control characters */
       /* except LF, BS, and DEL are ignored. If BS or DEL is encountered, a        */
       /* backspace is done.                                                         */

2          status = 0;
2          i, ch = 0;
2          do while (ch <> CR) and (ch <> LF) and (i < 198);
3              ch = c$in and 07FH;
3              if (ch = BS) or (ch = DEL) then
3              do;
```

292010-69

186/586 High Integration Board Initialization Routine (Continued)

```
            if i > 0 then
            do;
                i = i - 1;
                call c$out(DEL);
                call c$out(BS);
                call c$out(SP);
                call c$out(DEL);
                call c$out(BS);
            end;
            else
                call c$out(BEL);
        end;
        else
            if ch >= SP then
            do;
                call c$out(ch);
                buf(i) = ch;
                i = i + 1;
            end;
        else
            if (ch = CR) or (ch = LF) then
            do;
                buf(i) = CR;
                buf(i + 1) = LF;
                i = i + 2;
            end;
        else
            call c$out(BEL);
    end;
    call c$out(CR);
    if i > count then i = count;
    actual = i;
    do i = 0 to actual - 1;
        msg(i) = buf(i);
    end;

end read;


csts: procedure byte public;

    return ((input(SCC_CH_B_CMD) and 1) <> 0);

end csts;


write: procedure (file$id, msg$ptr, count, status$ptr) public;
declare (file$id, count) word,
        (msg$ptr, status$ptr) pointer,
        msg based msg$ptr (1) byte,
        status based status$ptr word,
        ch byte,
        i word;

/* This procedure implements the ISIS write */

    status = 0;
```

292010-70

186/586 High Integration Board Initialization Routine (Continued)

```
106   2         i = 0;
107   2         do while i < count;
108   3             ch = msg(i);
109   3             if ((ch >= SP) and (ch < DEL)) or (ch = CR) or (ch = LF) or (ch = NUL)
                    then
110   3                 call c$out(ch);
111   3             else
                        call c$out(QM);
112   3             i = i + 1;
113   3         end;

114   2     end write;


115   1     hib_ir:

            $IF EPROM27128

                    output(UMCS_reg) = 0F038H;    /* Starting Address = 0F0000H,
                                                     no wait state */

            $ELSE

                    output(UMCS_reg) = 0E038H;    /* Starting Address = 0E0000H,
                                                     no wait state */

            $ENDIF

116   1             output(LMCS_reg) = 03FCH;     /* 16K, no wait state */
117   1             output(PACS_reg) = 083CH;     /* PBA = 8000H, no wait state for
                                                     PSC0-3 */
118   1             output(MPCS_reg) = 0BFH;      /* Peripherals in I/O space, no A1 & A2
                                                     provided, 3 wait states for PSC4-6 */

119   1             call init$int$clt;
120   1             call init$SCC$B;
121   1             go to main;

122   1     end ini186;
```

292010-71

186/586 High Integration Board Initialization Routine (Continued)

APPENDIX C THE 82530 SCC - 80186 INTERFACE AP BRIEF

INTRODUCTION

The object of this document is to give the 82530 system designer an in-depth worst case design analysis of the typical interface to a 80186 based system. This document has been revised to include the new specifications for the 6 MHz 82530. The new specifications yield better margins and a 1 wait state interface to the CPU (2 wait states are required for DMA cycles). These new specifications will appear in the 1987 data sheet and advanced specification information can be obtained from your local Intel sales office. The following analysis includes a discussion of how the interface TTL is utilized to meet the timing requirements of the 80186 and the 82530. In addition, several optional interface configurations are also considered.

INTERFACE OVERVIEW

The 82530 - 80186 interface requires the TTL circuitry illustrated in Figure 1. Using five 14 pin TTL packages, 74LS74, 74AS74, 74AS08, 74AS04, and 74LS32, the following operational modes are supported:

- Polled
- Interrupt in vectored mode
- Interrupt in non-vectored mode
- Half-duplex DMA on both channels
- Full-duplex DMA on channel A

A brief description of the interface functional requirements during the five possible BUS operations follows below.

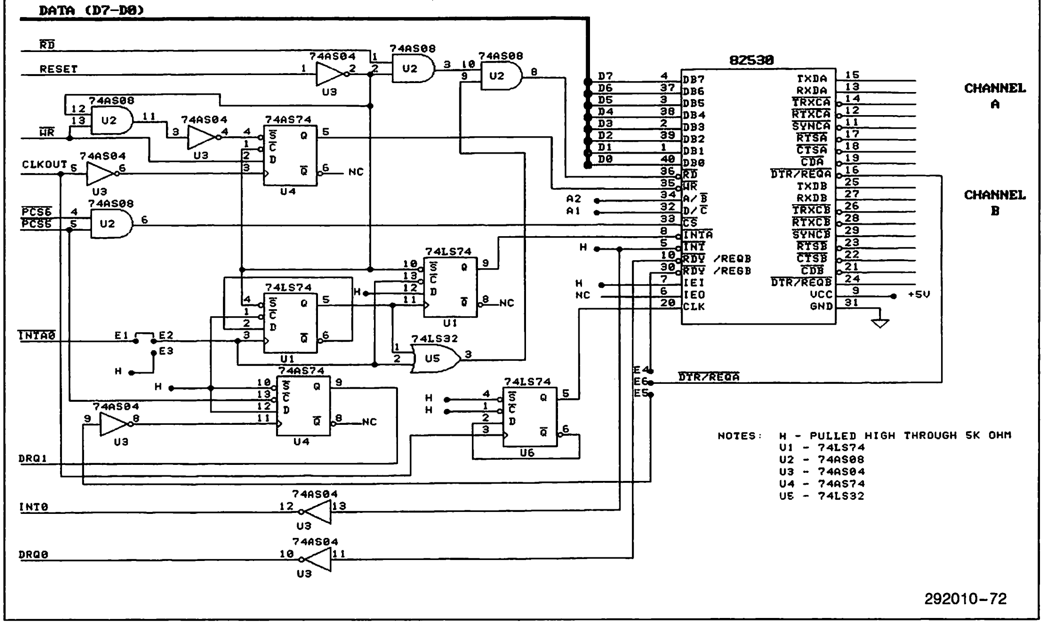


Figure 1. 82530-80186 Interface

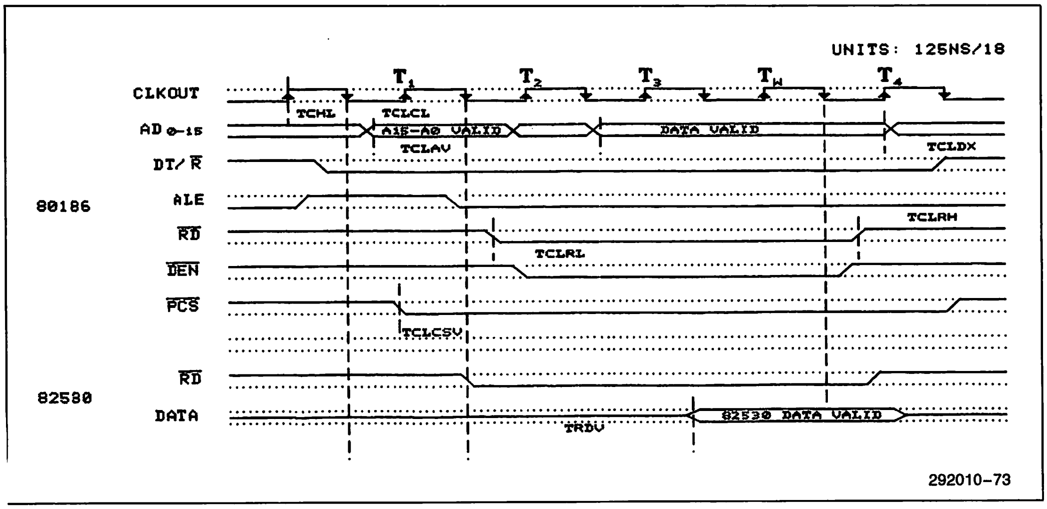


Figure 2. 80186-82530 Interface Read Cycle

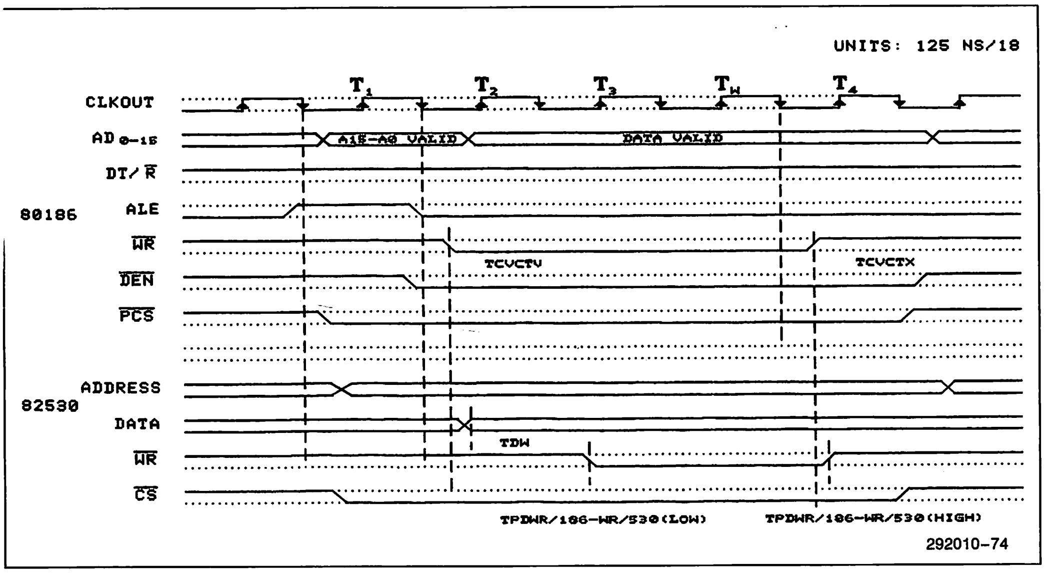


Figure 3. 80186-82530 Interface Write Cycle

READ CYCLE: The 80186 read cycle requirements are met without any additional logic, Figure 2. At least one wait state is required to meet the 82530 tAD access time.

WRITE CYCLE: The 82530 requires that data must be valid while the $\overline{\text{WR}}$ pulse is low, Figure 3. A D Flip-Flop delays the leading edge of $\overline{\text{WR}}$ until the falling edge of CLOCKOUT when data is guaranteed valid and $\overline{\text{WR}}$ is guaranteed active. The CLOCKOUT signal is inverted to assure that $\overline{\text{WR}}$ is active low before the D Flip-Flop is clocked. No wait states are necessary to meet the 82530's $\overline{\text{WR}}$ cycle requirements, but one is assumed from the $\overline{\text{RD}}$ cycle.

INTA CYCLE: During an interrupt acknowledge cycle, the 80186 provides two $\overline{\text{INTA}}$ pulses, one per bus cycle, separated by two idle states. The 82530 expects only one long $\overline{\text{INTA}}$ pulse with a $\overline{\text{RD}}$ pulse occurring only after the 82530 $\overline{\text{IEI}}/\overline{\text{IEO}}$ daisy chain settles. As

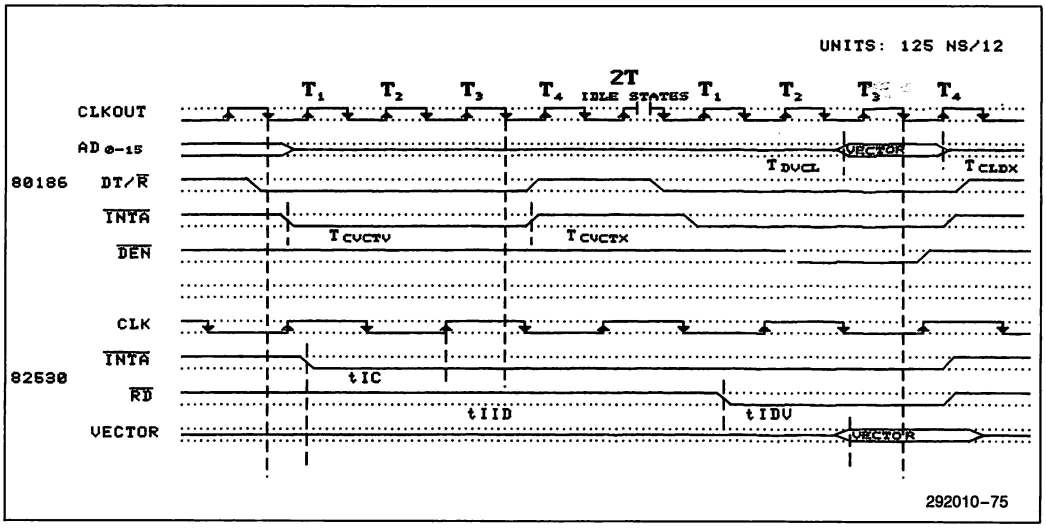


Figure 4. 82530–80186 $\overline{\text{INTA}}$ Cycle

illustrated in Figure 4, the $\overline{\text{INTA}}$ signal is sampled on the rising edge of CLK (82530). Two D Flip-Flops and two TTL gates, U2 and U5, are implemented to generate the proper $\overline{\text{INTA}}$ and $\overline{\text{RD}}$ pulses. Also, the $\overline{\text{INT}}$ signal is passively pulled high, through a 1 k resistor, and inverted through U3 to meet the 80186's active high requirement.

DMA CYCLE: Conveniently, the 80186 DMA cycle timings are the same as generic read and write operations. Therefore, with two wait states, only two modifications to the DMA request signals are necessary. First, the $\overline{\text{RDYREQA}}$ signal is inverted through U3 similar to the INT signal, and second the $\overline{\text{DTR/REQA}}$ signal is conditioned through a D Flip-Flop to prevent inadvertent back to back DMA cycles. Because the 82530 $\overline{\text{DTR/REQA}}$ signal remains active low for over five CLK (82530)'s, an additional DMA cycle could occur. This uncertain condition is corrected when U4 resets the $\overline{\text{DTR/REQ}}$ signal inactive high. Full Duplex on both DMA channels can easily be supported with one extra D Flip-Flop and an inverter.

RESET: The 82530 does not have a dedicated RESET input. Instead, the simultaneous assertion of both $\overline{\text{RD}}$ and $\overline{\text{WR}}$ causes a hardware reset. This hardware reset is implemented through U2, U3, and U4.

ALTERNATIVE INTERFACE CONFIGURATIONS

Due to its wide range of applications, the 82530 interface can have many varying configurations. In most of these applications the supported modes of operation need not be as extensive as the typical interface used in this analysis. Two alternative configurations are discussed below.

8288 BUS CONTROLLER: An 80186 based system implementing an 8288 bus controller will not require the preconditioning of the $\overline{\text{WR}}$ signal through the D Flip-Flop U4. When utilizing an 8288, the control signal $\overline{\text{IOWC}}$ does not go active until data is valid, therefore, meeting the timing requirements of the 82530. In such a configuration, it will be necessary to logically OR the $\overline{\text{IOWC}}$ with reset to accommodate a hardware reset operation.

NON-VECTORED INTERRUPTS: If the 82530 is to be operated in the non-vectored interrupt mode (B step only), the interface will not require U1 or U5. Instead, $\overline{\text{INTA}}$ on the 82530 should be pulled high, and pin 3 of U2 ($\overline{\text{RD}}$ AND $\overline{\text{RESET}}$) should be fed directly into the $\overline{\text{RD}}$ input of the SCC.

Obviously, the amount of required interface logic is application dependent and in many cases can be considerably less than required by the typical configuration, supporting all modes of SCC operation.

DESIGN ANALYSIS

This design analysis is for a typical microprocessor system, pictured in Figure 5. The Timing analysis assumes an 8 MHz 80186 and a 4 MHz 82530. Also, included in the analysis are bus loading, and TTL-MOS compatibility considerations.

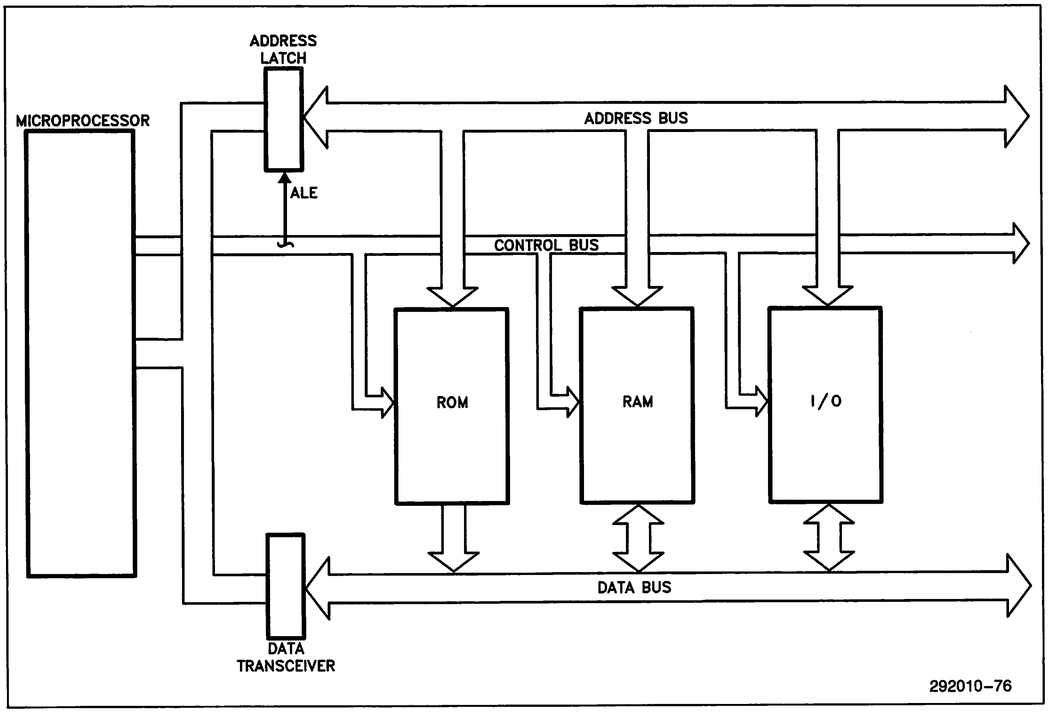


Figure 5. Typical Microprocessor System

Bus Loading and Voltage Level Compatabilities

The data and address lines do not exceed the drive capability of either 80186 or the 82530. There are several control lines that drive more than one TTL equivalent input. The drive capability of these lines are detailed below.

$\overline{\text{WR}}$: The $\overline{\text{WR}}$ signal drives U3 and U4.

* Iol (2.0 mA) > Iil (−0.4 mA + −0.5 mA)
Ioh (−400 μA) > Iih (20 μA + 20 μA)

$\overline{\text{PCS5}}$: The $\overline{\text{PCS5}}$ signal drives U2 and U4.

* Iol (2.0 mA) > Iil (−0.5 mA + −0.5 mA)
Ioh (−400 μA) > Iih (20 μA + 20 μA)

$\overline{\text{INTA}}$: The $\overline{\text{INTA}}$ signal drives 2(U1) and U5.

* Iol (2.0 mA) > Iil (−0.4 mA + −0.8 mA + −0.4 mA)
Ioh (−400 μA) > Iih (20 μA + 40 μA + 20 μA)

All the 82530 I/O pins are TTL voltage level compatible.

TIMING ANALYSIS

Certain symbolic conventions are adhered to throughout the analysis below and are introduced for clarity.

1. All timing variables with a lower case first letter are 82530 timing requirements or responses (i.e., tRR).
2. All timing variables with Upper case first letters are 80186 timing responses or requirements unless preceded by another device's alpha-numeric code (i.e., Tclcl or '373 Tpd).
3. In the write cycle analysis, the timing variable Tpd$\overline{\text{WR}}$186-$\overline{\text{WR}}$530 represents the propagation delay between the leading or trailing edge of the $\overline{\text{WR}}$ signal leaving the 80186 and the $\overline{\text{WR}}$ edge arrival at the 82530 $\overline{\text{WR}}$ input.

Read Cycle

1. **tAR:** Address valid to $\overline{\text{RD}}$ active set up time for the 82530. Since the propagation delay is the worst case path in the assumed typical system, the margin is calculated only for a propagation delay constrained and not an ALE limited path. The spec value is 0 ns minimum.

* 1 Tclcl − Tclav(max) − '245 Tpd(max) + Tclrl(min) + 2(U2) Tpd(min) − tAR(min)

= 125 − 55 − 20.8 + 10 + 2(2) − 0 = 63.2 ns margin

2. **tRA:** Address to $\overline{\text{RD}}$ inactive hold time. The ALE delay is the worst case path and the 82530 requires 0 ns minimum.

* 1 Tclcl − Tclrh (max) + Tchlh(min) + '373 LE Tpd(min) − 2(U2) Tpd(max)

= 55 − 55 + 5 + 8 − 2(5.5) = 2 ns margin

3. **tCLR:** $\overline{\text{CS}}$ active low to $\overline{\text{RD}}$ active low set up time. The 82530 spec value is 0 ns minimum.

* 1 Tclcl − Tclcsv(max) − Tclrl(min) − U2 skew($\overline{\text{RD}}$ − $\overline{\text{CS}}$) + U2 Tpd(min)

= 125 − 66 − 10 − 1 + 2 = 50 ns margin

4. **tRCS:** $\overline{\text{RD}}$ inactive to $\overline{\text{CS}}$ inactive hold time. The 82530 spec calls for 0 ns minimum.

* Tcscsx(min) − U2 skew($\overline{\text{RD}}$ − $\overline{\text{CS}}$) − U2 Tpd(max)

= 35 − 1 − 5.5 = 28.5 ns margin

5. **tCHR:** $\overline{\text{CS}}$ inactive to $\overline{\text{RD}}$ active set up time. The 82530 requires 5 ns minimum.

* 1 Tclcl + 1 Tchcl − Tchcsx(max) + Tclrl(min) − U2 skew ($\overline{\text{RD}}$ − $\overline{\text{CS}}$) + U2 Tpd(min) − tCHR

= 125 + 55 − 35 − 10 − 1 + 2 − 5 = 131 ns margin

6. **tRR:** $\overline{\text{RD}}$ pulse active low time. One 80186 wait state is included to meet the 150 ns minimum timing requirements of the 82530.

* Trlrh(min) + 1($\overline{\text{Tclcl}}$wait state) − 2(U2 skew) − tRR

= (250−50) + 1(125) − 2(1) − 150 = 173 ns margin

7. tRDV: $\overline{\text{RD}}$ active low to data valid maximum delay for 80186 read data set up time (Tdvcl = 20 ns). The margin is calculated on the Propagation delay path (worst case).

* 2 Tclcl + 1($\overline{\text{Tclcl}}$wait state) − Tclrl(max) − Tdvcl(min) − '245 Tpd(max) − 82530 tRDV(max) − 2(U2) Tpd(max)

= 2(125) + 1(125) − 70 − 20 − 14.2 − 105 − 2(5.5) = 154 ns margin

8. **tDF:** $\overline{\text{RD}}$ inactive to data output float delay. The margin is calculated to $\overline{\text{DEN}}$ active low of next cycle.

* 2 Tclcl + Tclch(min) − Tclrh(max) + Tchctv(min) − 2(U2) Tpd(max) − 82530 tDF(max)

= 250 + 55 −55 + 10 − 11 − 70 = 179 ns margin

9. **tAD:** Address required valid to read data valid maximum delay. The 82530 spec value is 325 ns maximum.

* 3 Tclcl + 1($\overline{\text{Tclcl}}$wait state) − Tclav(max) − '373 Tpd(max) − '245 Tpd − Tdvcl(min) − tAD

= 375 + 125 − 55 − 20.8 −14.2 − 20 −325 = 65 ns margin

Write Cycle

1. **tAW:** Address required valid to $\overline{\text{WR}}$ active low set up time. The 82530 spec is 0 ns minimum.

* Tclcl − Tclav(max) − Tcvctv(min) − '373 Tpd(max) + Tpd$\overline{\text{WR}}$186 − $\overline{\text{WR}}$530(LOW) [Tclcl − Tcvctv(min) + U3 Tpd(min) + U4 Tpd(min)] − tAW

= 125 − 55 − 5 − 20.8 + [125 − 5 + 1 + 4.4] − 0 = 170.6 ns margin

2. **tWA:** $\overline{\text{WR}}$ inactive to address invalid hold time. The 82530 spec is 0 ns.

* Tclch(min) − Tcvctx(max) + Tchlh(min) + '373 LE Tpd(min) − Tpd$\overline{\text{WR}}$186=$\overline{\text{WR}}$530(HIGH) [U2 Tpd(max) + U3 Tpd(max) + U4 Tpd(max)]

= 55 − 55 + 5 + 8 − [5.5 + 3 + 7.1] = −2.6 ns margin

3. **tCLW:** Chip select active low to $\overline{\text{WR}}$ active low hold time. The 82530 spec is 0 ns.

* 1 Tclcl − Tclcsv(max) + Tcvctv(min) − U2 Tpd(max) + Tpd$\overline{\text{WR}}$186=$\overline{\text{WR}}$530(LOW) [Tclcl − Tcvctv(min) + U3 Tpd(min) + U4 Tpd(min)]

= 125 − 66 + 5 − 5.5 + [125 − 5 + 1 + 4.4] = 183.9 ns margin

4. **tWCS:** $\overline{\text{WR}}$ invalid to Chip Select invalid hold time. 82530 spec is 0 ns.

* Tcxcsx(min) − U2 Tpd(max) − Tpd$\overline{\text{WR}}$186=$\overline{\text{WR}}$530(HIGH) [U2 Tpd(max) + U3 Tpd(max) + U4 Tpd(max)]

= 35 + 1.5 − [5.5 + 3 + 7.1] = 20.9 ns margin

5. **tCHW:** Chip Select inactive high to $\overline{\text{WR}}$ active low set up time. The 82530 spec is 5 ns.

* 1 Tclcl + Tchcl(min) + Tcvctv(min) − Tchcsx(max) − U2 Tpd(max) + Tpd$\overline{\text{WR}}$186=$\overline{\text{WR}}$530(LOW) [Tclcl − Tcvctv(min) + U3 Tpd(min) + U4 Tpd(min)] − tCHW

= 125 + 55 + 5 − 35 − 5.5 + [125 −5 + 1 + 4.4] − 5 = 264 ns margin

6. **tWW:** $\overline{\text{WR}}$ active low pulse. 82530 requires a minimum of 60 ns from the falling to the rising edge of $\overline{\text{WR}}$. This includes one wait state.

* Twlwh [2Tclcl − 40] + 1 (Tclclwait state) − Tpd$\overline{\text{WR}}$/186−$\overline{\text{WR}}$530(LOW) [Tclcl − Tcvctv(min) + U3 Tpd(max) + U4 Tpd(max)] + Tpd$\overline{\text{WR}}$/186=$\overline{\text{WR}}$/530(HIGH) [U2 Tpd(min) U3 Tpd(min) + U4 Tpd(min)] − tWW

= 210 + 1(125) − [125 − 5 + 4.5 + 9.2] − [1.5 + 1 + 3.2] − 60 = 135.6 ns margin

7. **tDW:** Data valid to $\overline{\text{WR}}$ active low setup time. The 82530 spec requires 0 ns.

* Tcvctv(min) − Tcldv(max) − '245 Tpd(max) + Tpd$\overline{\text{WR}}$186−$\overline{\text{WR}}$530(LOW) [Tclcl − Tcvctv(min) + U3 Tpd(min) + U4 Tpd(min)]

= 5 − 44 − 14.2 + 125 − 5 + 1.0 + 4.4 = 72.2 ns margin

8. **tWD:** Data valid to $\overline{\text{WR}}$ inactive high hold time. The 82530 requires a hold time of 0 ns.

* Tclch − skew {Tcvctx(max) + Tcvctx(min)} + '245 OE Tpd(min) − Tpd$\overline{\text{WR}}$186−$\overline{\text{WR}}$530(HIGH) [U2 Tpd(max) + U3 Tpd(max) + U4 Tpd(max)]

= 55 − 5 + 11.25 − [5.5 + 3.0 + 7.1] = −50.6 ns margin

INTA Cycle:

1. **tIC:** This 82530 spec implies that the $\overline{\text{INTA}}$ signal is latched internally on the rising edge of CLK (82530). Therefore the maximum delay between the 80186 asserting $\overline{\text{INTA}}$ active low or inactive high and the 82530 internally recognizing the new state of $\overline{\text{INTA}}$ is the propagation delay through U1 plus the 82530 CLK period.

* U1 Tpd(max) + 82530 CLK period

= 45 + 250 = 295 ns

2. **tCI:** rising edge of CLK to $\overline{\text{INTA}}$ hold time. This spec requires that the state of $\overline{\text{INTA}}$ remains constant for 100 ns after the rising edge of CLK. If this spec is violated any change in the state of $\overline{\text{INTA}}$ may not be internally latched in the 82530. tCI becomes critical at the end of an $\overline{\text{INTA}}$ cycle when $\overline{\text{INTA}}$ goes inactive. When calculating margins with tCI, an extra 82530 CLK period must be added to the $\overline{\text{INTA}}$ inactive delay.

3. **tIW:** $\overline{\text{INTA}}$ inactive high to $\overline{\text{WR}}$ active low minimum setup time. The spec pertains only to 82530 $\overline{\text{WR}}$ cycle and has a value of 55 ns. The margin is calculated assuming an 82530 $\overline{\text{WR}}$ cycle occurs immediately after an $\overline{\text{INTA}}$ cycle. Since the CPU cycles following an 82530 $\overline{\text{INTA}}$ cycle are devoted to locating and executing the proper interrupt service routine, this condition should never exist. 82530 drivers should insure that at least one CPU cycle separates $\overline{\text{INTA}}$ and $\overline{\text{WR}}$ or $\overline{\text{RD}}$ cycles.

4. **tWI:** $\overline{\text{WR}}$ inactive high to $\overline{\text{INTA}}$ active low minimum hold time. The spec is 0 ns and the margin assumes CLK coincident with $\overline{\text{INTA}}$.

* Tclcl − Tcvctx(max) − Tpd$\overline{\text{WR}}$186 − $\overline{\text{WR}}$530(HIGH) [U3 Tpd(max) + U4 Tpd(max)] + Tcvctv(min) + U1 Tpd(min)

= 125 − 55 − [5.5 + 3 + 7.1] + 5 + 10 = 69.4 ns margin

5. **tIR:** $\overline{\text{INTA}}$ inactive high to $\overline{\text{RD}}$ active low minimum setup time. This spec pertains only to 82530 $\overline{\text{RD}}$ cycles and has a value of 55 ns. The margin is calculated in the same manner as tIW.

6. **tRI:** $\overline{\text{RD}}$ inactive high to $\overline{\text{INTA}}$ active low minimum hold time. The spec is 0 ns and the margin assumes CLK coincident with $\overline{\text{INTA}}$.

* Tclcl − Tclrh(max) − 2 U2 Tpd(max) + Tcvctv(min) + U1 Tpd(min)

= 125 − 55 − 2(5.5) + 5 + 10 = 74 ns margin

7. **tIID:** $\overline{\text{INTA}}$ active low to $\overline{\text{RD}}$ active low minimum setup time. This parameter is system dependent. For any SCC in the daisy chain, tIID must be greater than the sum of tCEQ for the highest priority device in the daisy chain, tEI for this particular SCC, and tEIEO for each device separating them in the daisy chain. The typical system with only 1 SCC requires tIID to be greater than tCEQ. Since tEI occurs coincidently with tCEQ and it is smaller it can be neglected. Additionally, tEIEO does not have any relevance to a system with only one SCC. Therefore tIID > tCEQ = 250 ns.

* 4 Tclcl + 2 Tidle states − Tcvctv(max) − tIC [U1 Tpd(max) + 82530 CLK period] + Tcvctv(min) + U5 Tpd(min) + U2 Tpd(min) − tIID

= 500 + 250 − 70 − [45 + 250] + 5 + 6 + 2 − 250 = 148 ns margin

8. **tIDV:** $\overline{\text{RD}}$ active low to interrupt vector valid delay. The 80186 expects the interrupt vector to be valid on the data bus a minimum of 20 ns before T4 of the second acknowledge cycle (Tdvcl). tIDV spec is 100 ns maximum.

* 3 Tclcl − Tcvctv(max) − U5 Tpd(max) − U2 Tpd(max) − tIDV(max) − '245 Tpd(max) − Tdvcl(min)

= 375 − 70 − 25 − 5.5 − 100 − 14.2 − 20 = 140.3 ns margin

9. **tll:** $\overline{\text{RD}}$ pulse low time. The 82530 requires a minimum of 125 ns.

* 3 Tclcl − Tcvctv(max) − U5 Tpd(max) − U2 Tpd(max) + Tcvctx(min) + U5 Tpd(min) + U2 Tpd(min) − tll(min)

= 375 − 70 − 25 − 5.5 + 5 + 6 + 1.5 − 125 = 162 ns margin

DMA Cycle

Fortunately, the 80186 DMA controller emulates CPU read and write cycle operation during DMA transfers. The DMA transfer timings are satisfied using the above analysis. Because of the 80186 DMA request input requirements, two wait states are necessary to prevent inadvertent DMA cycles. There are also $\overline{\text{CPU}}$DMA intracycle timing considerations that need to be addressed.

1. **tDRD:** $\overline{\text{RD}}$ inactive high to $\overline{\text{DTRREQ}}$ (REQUEST) inactive high delay. Unlike the $\overline{\text{READYREQ}}$ signal, $\overline{\text{DTRREQ}}$ does not immediately go inactive after the requested DMA transfer begins. Instead, the $\overline{\text{DTRREQ}}$ remains active for a maximum of 5 tCY + 300 ns. This delayed request pulse could trigger a second DMA transfer. To avoid this undesirable condition, a D Flip Flop is implemented to reset the $\overline{\text{DTRREQ}}$ signal inactive low following the initiation of the requested DMA transfer. To determine if back to back DMA transfers are required in a source synchronized configuration, the 80186 DMA controller samples the service request line 25 ns before T1 of the deposit cycle, the second cycle of the transfer.

* 4 Tclcl − Tclcsv(max) − U4Tpd(max) − Tdrqcl(min)

= 500 − 66 − 10.5 − 25 = 398.5 ns margin

2. tRRI: 82530 $\overline{\text{RD}}$ active low to $\overline{\text{REQ}}$ inactive high delay. Assuming source synchronized DMA transfer, the 80186 requires only one wait state to meet the tRRI spec of 200 ns. Two are included for consistency with tWRI.

* 2 Tclcl + 2($\overline{\text{Tclcl}}$wait state) − Tclrl(max) − 2(U2) Tpd(max) − Tdrqcl − tRRI

=2(125) + 2(125) − 70 − 2(5.5) − 200 = 219 ns margin

3. tWRI: 82530 $\overline{\text{WR}}$ active low to $\overline{\text{REQ}}$ inactive high delay. Assuming destination synchronized DMA transfers, the 80186 needs two wait states to meet the tWRI spec. This is because the 80186 DMA controller samples requests two clocks before the end of the deposit cycle. This leaves only 1 Tclcl + n(wait states) minus $\overline{\text{WR}}$ active delay for the 82530 to inactivate its $\overline{\text{REQ}}$ signal.

* Tclcl + 2($\overline{\text{Tclcl}}$wait state) − Tcvctv(min) − Tpd$\overline{\text{WR}}$186−$\overline{\text{WR}}$530(LOW) [Tclcl − Tcvctv(min) + U3 Tpd(max) + U4 Tpd(max)] − Tdrqcl − tWRI

=375 − 5 − [125 − 5 + 4.5 + 9.2] − 25 − 200 = 11.3 ns margin

NOTE:
If one wait state DMA interface is required, external logic, like that used on the $\overline{\text{DTRREQ}}$ signal, can be used to force the 82530 $\overline{\text{REQ}}$ signal inactive.

4. **tREC:** CLK recovery time. Due to the internal data path, a recovery period is required between SCC bus transactions to resolve metastable conditions internal to the SCC. The DMA request lines are marked from requesting service until after the tREC has elapsed. In addition, the CPU should not be allowed to violate this recovery period when interleaving DMA transfers and CPU bus cycles. Software drivers or external logic should orchestrate the CPU and DMA controller operation to prevent tREC violation.

Reset Operation

During hardware reset, the system RESET signal is asserted high for a minimum of four 80186 clock cycles (1000 ns). The 82530 requires $\overline{\text{WR}}$ and $\overline{\text{RD}}$ to be simultaneously asserted low for a minimum of 250 ns.

* 4 Tclcl − U3 Tpd(max) − 2(U2) Tpd(max) + U4 Tpd(min) − tREC

= 1000 − 17.5 − 2(5.5) + 3.5 − 250 ns = 725 ns margin

INTEL CORPORATION, 3065 Bowers Ave., Santa Clara, CA 95051; Tel. (408) 987-8080

INTEL CORPORATION (U.K.) Ltd., Swindon, United Kingdom; Tel. (0793) 696 000

Printed in England by Ben Johnson/3K/860525