CIMETRICS TECHNOLOGY

NBS-10 NINE BIT SERIAL CARD

USER'S MANUAL

CIMETRICS TECHNOLOGY
55 TEMPLE PLACE
BOSTON, MA 02111-1300
U.S.A.

TELEPHONE: (617) 350-7550
FAX: (617) 350-7552
EMAIL: info@cimetrics.com
URL: http://www.cimetrics.com

LIMITED WARRANTY: NBS-10 HARDWARE

Cimetrics Technology warrants the NBS-10 card against defects in materials and workmanship for a period of Ninety (90) days from the date of purchase.

If you discover a defect, Cimetrics Technology will, at its option, repair, replace, or refund the purchase price of the product at no charge to you, provided you return it during the warranty period, transportation charges prepaid, to Cimetrics Technology. Please attach your name, phone number, a description of the problem and a copy of the bill of sale bearing the appropriate Cimetrics Technology serial numbers as proof of date of original purchase, to each product returned for warranty service.

This warranty does not apply if the product has been damaged by accident, abuse or misuse, or misapplication, has been modified without the written permission of Cimetrics Technology, or if any serial number has been removed or defaced.

Some states do not allow the exclusion or limitation of implied warranties or liability for incidental and consequential damages, so the above limitation or exclusion may not apply to you. This warranty gives you specific legal rights, and you may also have other rights which vary from state to state.

LIABILITY DISCLAIMER

Cimetrics Technology assumes no liability for damages, lost profits, lost savings, lost goodwill, downtime, or any other incidental or consequential damage resulting from the use, misuse of, or inability to use this product. Cimetrics Technology will not be liable for any claim made by any other related party. No Cimetrics Technology dealer, agent, or employee is authorized to make any modification, extension or addition to this policy.

Cimetrics Technology's products are not authorized for use as critical components in life support devices or systems.

DOCUMENTATION DISCLAIMER

Cimetrics Technology makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Cimetrics shall not be liable for errors contained herein or for incidental consequential damages in connection with the furnishing, performance, or use of this material.

The information contained within this document is subject to change without notice and does not represent a commitment on the part of Cimetrics Technology.

Table of Contents

OVERVIEW

INSTALLATION

Table of Contents

PROGRAMMING THE NBS-10

Appendix A

1 OVERVIEW

1.1 DESCRIPTION OF THE NBS-10

The NBS-10 is a flexible RS-485/422 special purpose asynchronous serial adapter for PC/XT/AT compatible computers. Based on the Intel 82510 UART, the NBS-10 provides several features particularly suited to RS-485 multidrop networking. The NBS-10 can also be used as an RS-422 serial link.

FEATURES:

- Full- or Half-duplex 9-bit serial communication mode
- Half- or full-duplex standard PC communications
- Flexible control of transmit direction - RTS/DTR
- Jumper selectable termination resistors
- Removable bias resistors

1.2 AUDIENCE

To use this product, you should be familiar with PC compatible computers and DOS operating systems. This manual is written for engineers involved with personal computers and embedded systems networking.

1.3 ABOUT THIS MANUAL

This manual is organized as follows:

Chapter 1: Overview
Introduces the NBS-10 features and gives an overview of this manual.

Chapter 2: Installation
Provides detailed instructions and flowcharts on how to set up the NBS-10 for operation.

Chapter 3: Programming
Describes the operation of the NBS-10 in typical programming applications.

Appendix A
Sample Program

Appendix B
INTEL 82510 Data Sheets.

1.4 REFERENCES AND ADDITIONAL INFORMATION

Electronics Industry Association Publications

[1] "High Speed 25-Position Interface for Data Terminal Equipment and Data Circuit-Terminating Equipment" (EIA-530), Electronics Industry Association, 1987.

[2] "Standard for Electrical Characteristics of Generators and Receivers for Use in Balanced Digital Multipoint Systems"(EIA-485), Electronics Industry Association, 1983.

[3] "Electrical Characteristics of Balanced Voltage Digital Interface Circuits" (EIA-422-A), Electronics Industry Association, 1978.

PC Reference Material

[4] Eggebrecht, Lewis, Interfacing to the IBM Personal Computer, Howard W. Sams and Co., 1990 (Second Edition).

[5] System BIOS for IBM PC/XT/AT Computers and Compatibles, Phoenix Technologies Ltd., Addison-Wesley, 1989.

[6] Mueller, Scott, Upgrading and Repairing PCs, QUE Corporation, 1988.

[7] Duncan, Ray, Advanced MS-DOS Programming, Microsoft Press, 1988.

Data Books

[8] Interface Databook (1990), National Semiconductor Corp.

[9] Microcommunications, Intel Corporation, 1991

[10] Embedded Control Applications, Intel Corporation, 1991.

Networking/Microcontroller Networking

[11] Nisley, Ed, "A network for Distributed Control, Part 1," Circuit Cellar Ink, August/September 1989, pp. 32-39.

[12] The BITBUS Interconnect Serial Control Bus Specification, Intel Corp., 1988.

Nine-Bit Embedded Control Networking

[13] Woehr, Jack, "Multidrop Processing", Embedded Systems Programming, March 1990, pp. 58-67. Nine-data-bit communication using the 68681 DUART.

[14] Dhuse, Jon, and George R. Hayek, "Standard Protocols Are Needed for Distributed Microcontrollers," Data Communications, January 1986, pp. 171-175.

[15] Horden, Ira, and David Ryan, "Microcontrollers for Factory Automation," Machine Design, March 6, 1986, pp. 117-120. Nine-bit protocols using the 8096.

[16] Bruntlett, John E., "Serial Protocol for Distributed Microcontrollers," Wescon/86 Conference Record.

[17] Simmers, Chuck, "Specialized I/O and High Speed CPU Yields Efficient Microcontroller for Automotive Applications," IEEE CH2072-7/84/0000-0003.

[18] Butler, Jim, "A Simple RS-485 Network," Circuit Cellar Ink , June/July 1991, Issue 21.

[19] Butler, Jim, "Embedded Controller Networking Alternatives," Circuit Cellar Ink, April/May 1992, Issue 26.

1.5 FCC INFORMATION

WARNING: This equipment generates, uses, and can radiate radio frequency energy. If not installed and used in accordance with the instruction manual, it may cause interference to radio communications. This equipment has been tested and found to comply with the limits for a Class A computing device pursuant to Subpart B, Part 15 of the FCC Rules, which are designed to provide reasonable protection against such interference when operated in a commercial environment. Operation of this equipment in a residential area is likely to cause interference in which case the user at his own expense will be required to take whatever measures may be required to correct the interference.

The NBS-10 has been verified for FCC Part 15 Class A computing device compliance using shielded twisted-pair cable and shielded head shells. Your installation must use shielded components in order to satisfy the Part 15 requirements.

2 INSTALLATION

2.1 THE CONFIGURATION PROCESS

The installation material in this manual is designed with two groups in mind. For experienced users, configuration information is presented in flow charts and a series of tables, quickly enabling installation of the NBS-10. For those with more limited familiarity with personal computers and RS-485 networking a tutorial explaining RS-485/422 is included in addition to the charts (section 2.2). This section should provide the less-seasoned user with the background information necessary for continuing with installation.

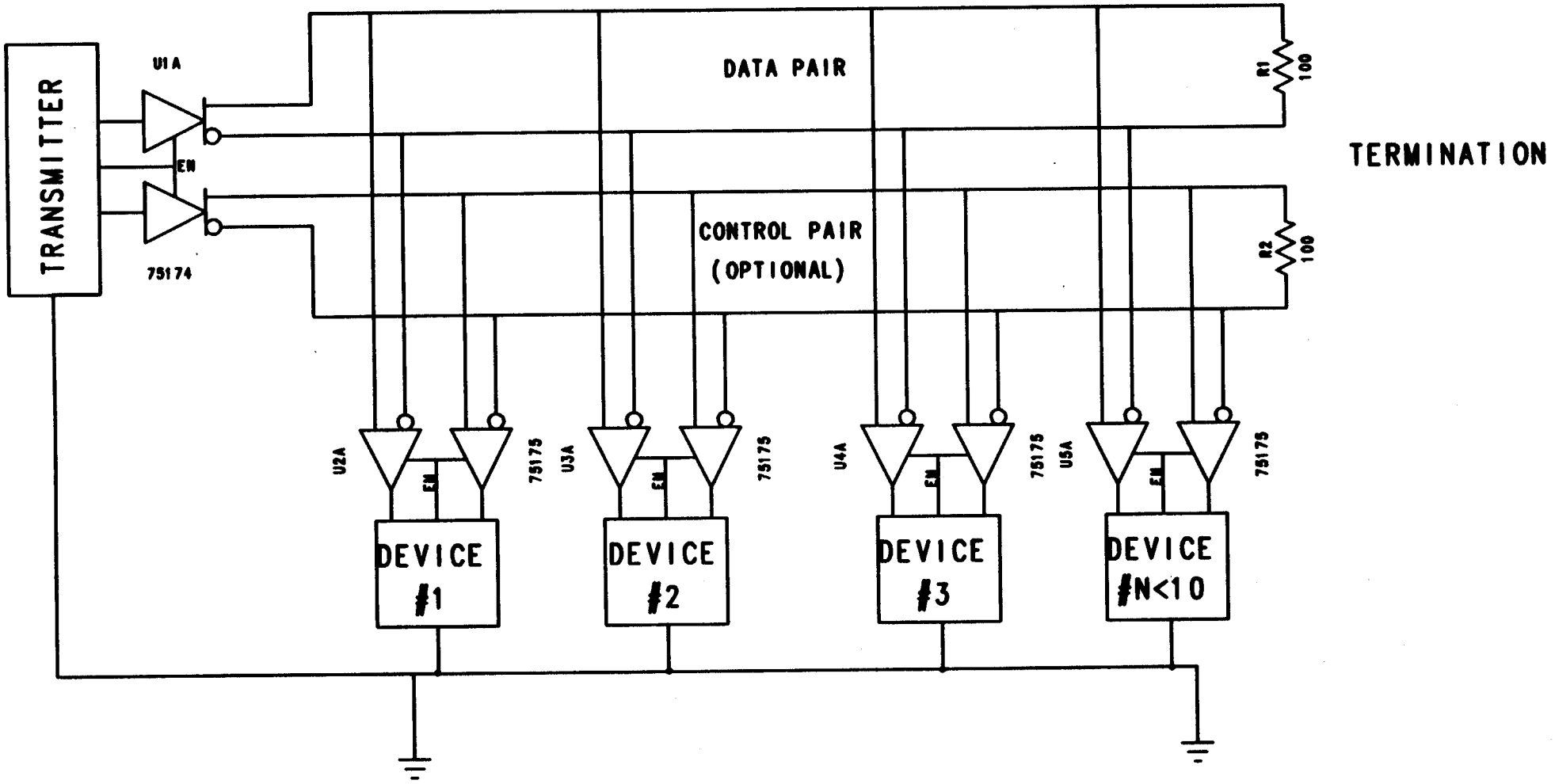
After the one-section introduction to RS-485/422, a worksheet is presented in order to facilitate organization of your configuration data (section 2.3). The worksheet is designed for use in conjunction with the flow chart (figure 2.4). Questions that might arise while using the flow chart are addressed in subsequent sections; relevant sections are indicated on the flow chart in parentheses (for example, "How do I know whether to use half- or full-duplex?", step four on the flow chart, is answered in section 2.6).

2.2 UNDERSTANDING RS-422 and RS-485

This section discusses standards: what they are and how they work. Over the years, a number of industry standards have been developed to cope with a broad range of communications problems; you are probably already familiar with the RS-232 standard in its many incarnations. For this reason, we will use the RS-232 as a basis for comparison throughout this section. This discussion will center around RS-422 and RS-485. Unlike the RS-232 standard, which includes specifications for both the electrical and mechanical properties of the interface problem, RS-422 and RS-485 are Electronics Industry Association (EIA) standards that describe and specify only the ELECTRICAL characteristics of that problem.

(Note: none of the above-mentioned standards recommends or specifies a protocol in any way. A communication protocol such as Cimetrics Technology's NSP must be used in conjunction with these standards.)

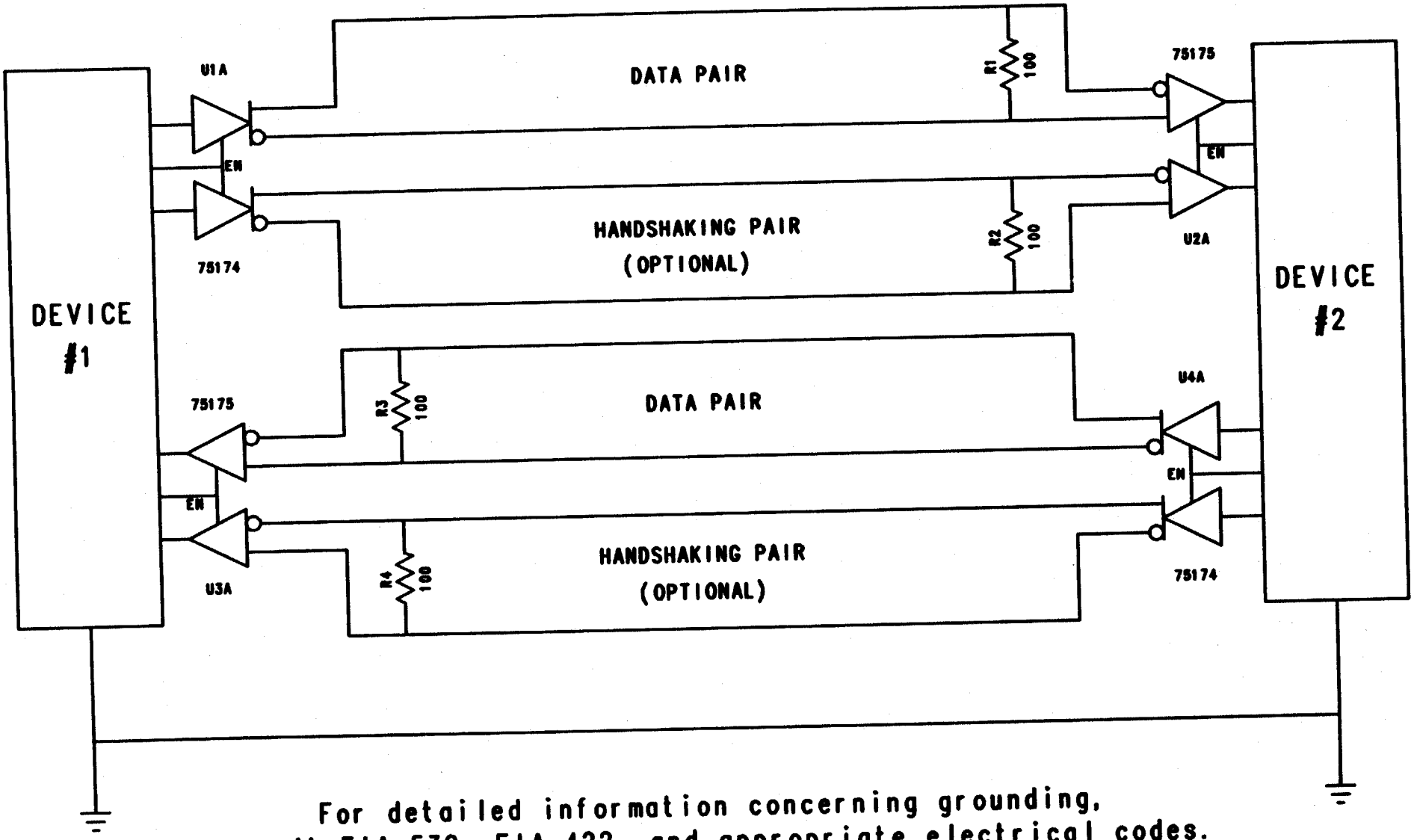
RS-422 MULTIPLE RECEIVER CONFIGURATION



For detailed information concerning grounding, consult EIA-530, EIA-422, and appropriate electrical codes.

FIGURE 2.1

RS-422/530 FULL DUPLEX CONFIGURATION



For detailed information concerning grounding, consult EIA-530, EIA-422, and appropriate electrical codes.

FIGURE 2.2

2.2.1 RS-422

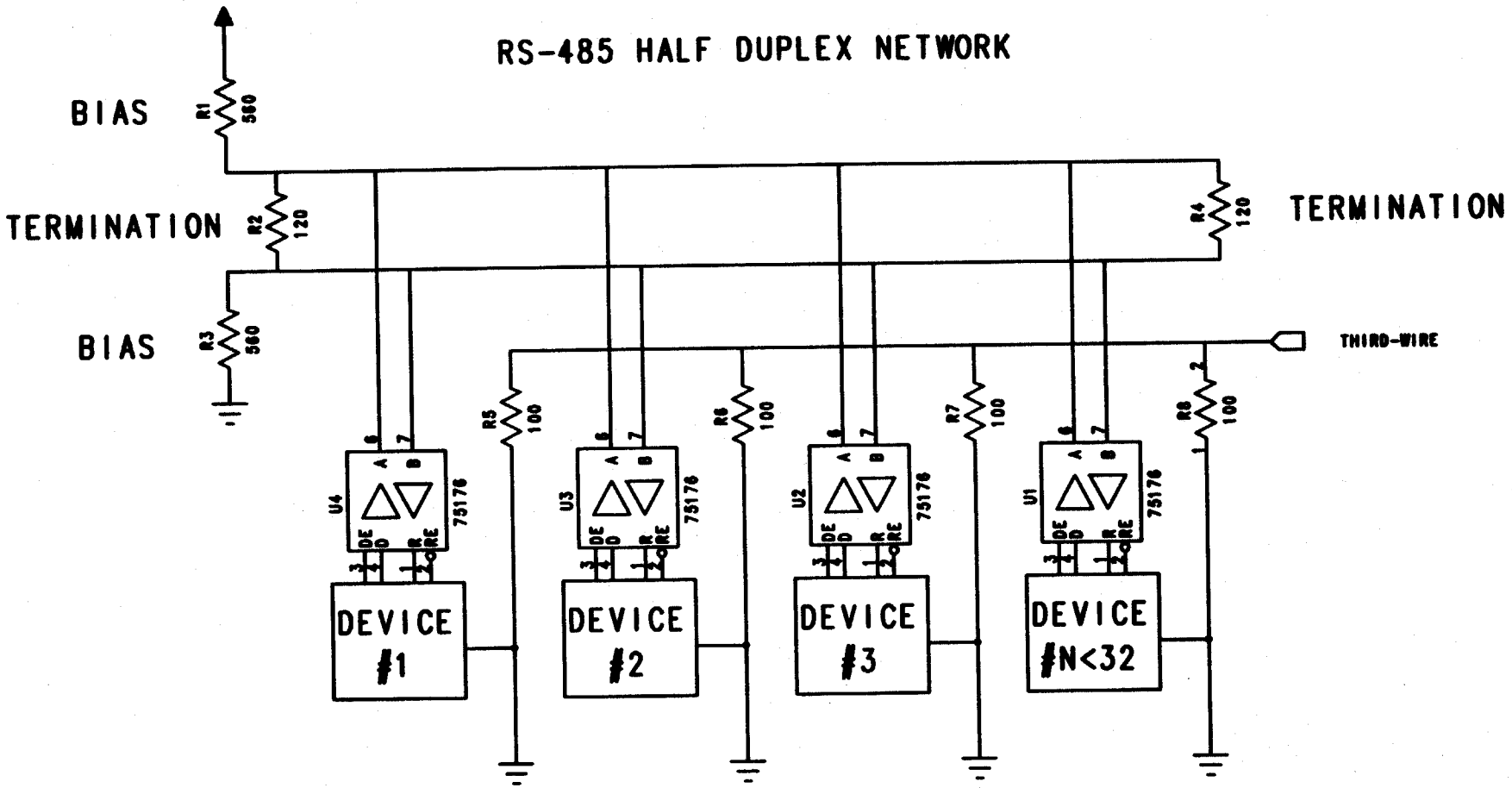
In 1975 the EIA introduced RS-422, a standard which uses differential (balanced) data transmission in one direction along a transmission line. A differential signal is represented by the voltage difference between two conductors of the transmission line (not by the voltage from each of the conductors to ground). In RS-422 the driver (transmitter) is at one end of the line with up to 10 receivers on the line and a 100-ohm termination resistor at the other end (see figure 2.1). The RS-422 standard is typically used in full duplex mode with one pair of transmission lines (usually twisted-pair wire) for sending and another pair for receiving data (see figure 2.2). In the configuration represented in figure 2.3, a 100-ohm termination resistor would be placed across the receiver at the far end of the line.

RS-422 offers greatly improved characteristics over RS-232 in noise immunity, speed, and distance capabilities. [3]

COMPARISON OF RS-232, RS-422, RS-485			
PARAMETER	**RS-232**	**RS-422**	**RS-485**
TRANSMISSION MODE	SINGLE-ENDED	DIFFERENTIAL	DIFFERENTIAL
MAX CABLE LENGTH	50'	4000'	4000'
MAX NUMBER DRIVERS	1	1	32
MAX NUMBER RECEIVERS	1	10	32
MAX DATA RATE	20K BITS/S	10M BITS/S	10M BITS/S
DRIVER Z_{OUT} POWER OFF	300 OHMS	60K OHMS	120K OHMS
RECEIVER LOAD IMP.	3K - 7K OHMS	>4K OHMS	>12K OHMS
RECEIVER SENSITIVITY	+/- 3V	+/- 200mV	+/- 200mV
LOAD IMPEDANCE	3K - 7K OHMS	100 OHMS	60 OHMS
COMMON MODE RANGE	+/- 25V	-0.25V to +6V	-7V to +12V

2.2.3 RS-485

In 1983, the EIA approved RS-485, a new differential transmission standard considered by many to be an extension of the existing RS-422 standard. RS-485 specifies the electrical characteristics of receivers and drivers which are intended to operate in a balanced multipoint or party-line configuration. The RS-485 network is designed to support 32 receivers and drivers operating over twisted-pair wire terminated at both ends by 120-ohm resistors. The resistors should be at the extreme ends and all nodes should be directly connected (daisy chained) to the network or connected to it with short stubs. Although 32 nodes can exist on the network, only one may transmit at any given time. If two or more nodes attempt to transmit at the same time, a collision will result, causing garbled data. The network receivers and drivers are designed to tolerate this fault condition for a limited amount of time, but the situation should be avoided. Proper operation of the RS-485 interface circuits requires a signal return path between circuit grounds of the devices. This signal return path may be provided by a third wire, or by connecting all devices to an earth reference. When this path is provided by a third wire, the wire should be connected to the device through some resistance (1002Ω in the NBS-10) in order to limit current. Care must be taken not to form ground loops. For detailed information concerning grounding, consult the RS-485 standard and appropriate local, national, and international electrical codes.



For detailed information concerning grounding, consult EIA-485 and appropriate electrical codes.

FIGURE 2.3

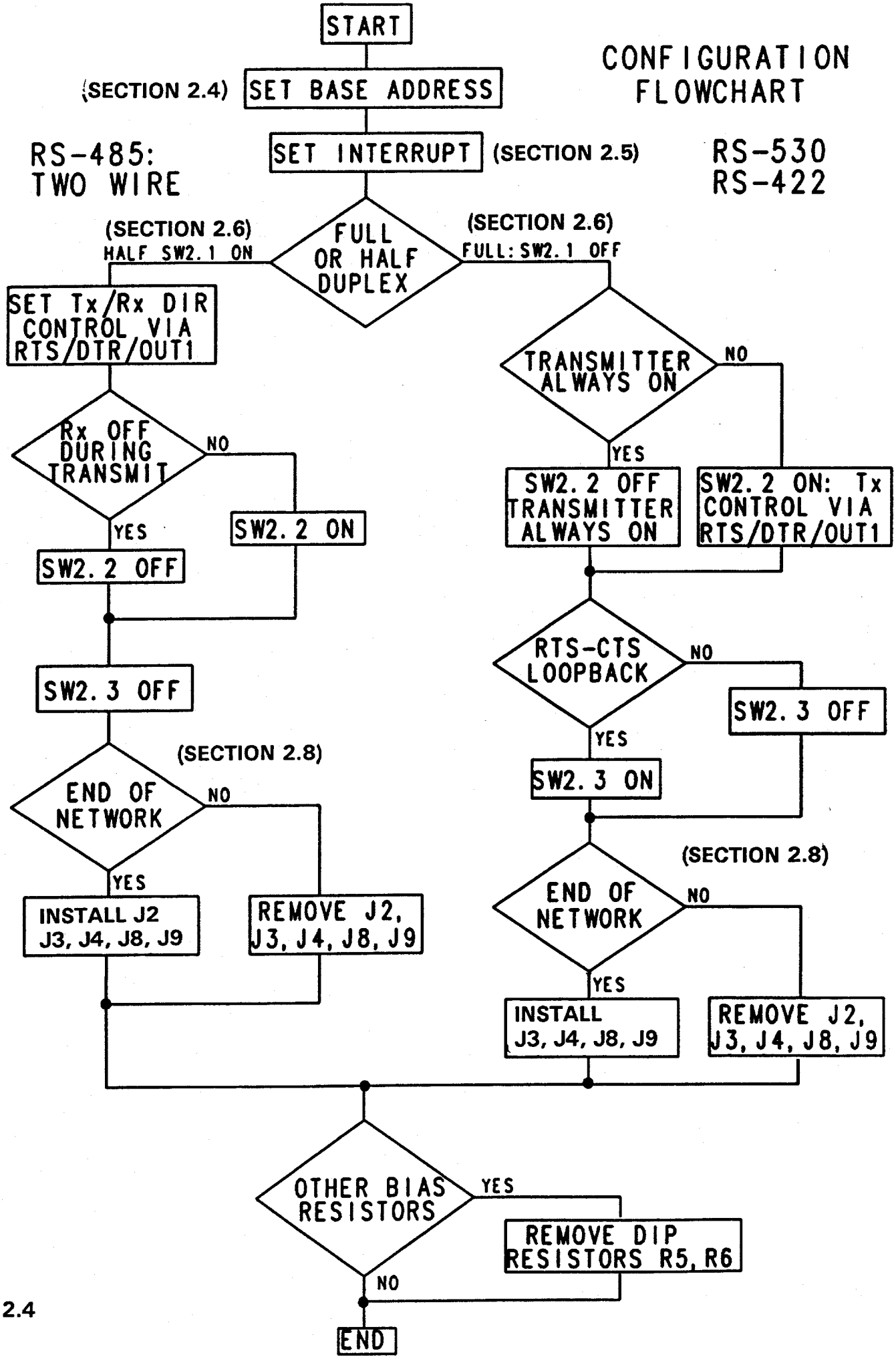

CONFIGURATION FLOWCHART
START
(SECTION 2.4) SET BASE ADDRESS
SET INTERRUPT (SECTION 2.5)
RS-485: TWO WIRE
RS-530 RS-422
FULL OR HALF DUPLEX
(SECTION 2.6) HALF SW2.1 ON
(SECTION 2.6) FULL: SW2.1 OFF
SET Tx/Rx DIR CONTROL VIA RTS/DTR/OUT1
Rx OFF DURING TRANSMIT
NO
YES
SW2.2 ON
SW2.2 OFF
SW2.3 OFF
(SECTION 2.8)
END OF NETWORK
NO
YES
INSTALL J2 J3, J4, J8, J9
REMOVE J2, J3, J4, J8, J9
TRANSMITTER ALWAYS ON
NO
YES
SW2.2 OFF TRANSMITTER ALWAYS ON
SW2.2 ON: Tx CONTROL VIA RTS/DTR/OUT1
RTS-CTS LOOPBACK
NO
YES
SW2.3 OFF
SW2.3 ON
(SECTION 2.8)
END OF NETWORK
NO
YES
INSTALL J3, J4, J8, J9
REMOVE J2, J3, J4, J8, J9
OTHER BIAS RESISTORS
YES
NO
REMOVE DIP RESISTORS R5, R6
END

FIGURE 2.4

2.2.4 UARTS AND BIAS RESISTORS

The RS-485 voltage standard as defined by the EIA does not specify the output level of a receiver when no input is applied to the line. The RS-485 threshold voltage between high and low logic levels is defined with a +/- 200mV indeterminant area around the nominal threshold of 0 volts (i.e., zero potential difference voltage between the two lines). When the transmission line is properly terminated, and no transmitters are activated, the voltage level on the transmission line usually floats in this indeterminant area. Any noise picked up on the line may modulate the tristated line in this threshold region, producing random data at the receiver.

This random information wrecks havoc with traditional UARTs. These UARTs are designed to interpret a line in the mark (high) state as an idle condition, unless good (non-random) data is available. The UARTs view the first one to zero transition as a start bit, normally used to synchronize the rest of the byte. Random noise appears as false start bits and data, causing the network to malfunction.

To get around this tristated line condition of RS-485, a differential bias can be applied to the transmission line. When no transmitters are on the line, this technique can be used to push the line out of the threshold area into the high state. Different biasing techniques have been successfully used on RS-485 lines. In a network where the length and characteristic impedance of the line are not known, DC biasing is the preferred method. A simple voltage divider is often used to hold the differential line in a state which produces a one at the output of the receiver.

By the voltage divider rule we know that the voltage at V1 is given by:

$$V1 = \frac{(R2+RT)}{R1+(R2+RT)} Vcc = \frac{(620)}{1180} 5V = 2.627V$$

By symmetry, we know V2 is 2.373V

Therefore V_{diff} = 2.627V - 2.373V = 254mV

Thus the differential pair is held apart by 254mV with the non-inverting line more positive than the inverting line.

Most RS-485 serial products which employ UARTs provide a DC bias resistor network on the card. When one has more than one card on a network, however, the presence of more than one bias source is problematic for several reasons:

Additional bias networks affect the impedance of the transmission line;
Bias networks often do not have the same polarities or voltage levels, especially if the cards are produced by different manufacturers.

The Cimetrics Technology NBS-10 is designed with removable bias resistor networks so that if another bias source already exists, the Cimetrics bias networks can be removed from the card.

Termination resistors should be placed only at the ends of the transmission line. There should be ONLY TWO SUCH RESISTORS ON THE ENTIRE NETWORK. More than two will substantially lower the impedance of the line and will hinder the line drivers.

The most common configuration for RS-485 in multidrop embedded control networking is the half-duplex, two-wire system (see figure 2.3).

2.3 NBS-10 CONFIGURATION WORKSHEET

Before you get started on the NBS-10 configuration process, it's a good idea to organize information about your application. By filling out the following worksheet, and then following the adjacent flowchart, you should be able to complete the configuration in a minimum amount of time.

(1) What base address should be assigned to the NBS-10? The NBS-10 requires eight consecutive locations above the base address.

(2) What interrupt will the NBS-10 use?

(3) For what application will the NBS-10 be used?

(A) RS-422 communications
(B) RS-485 networking/embedded control networking

(4) Will the card be used in full or half duplex?

(5) If half duplex:

(A) What should control the direction of transmission? (RTS or DTR)
(B) Should receiver be off during transmit?

(6) If full duplex:

(A) Should the transmitter always be on?
If no, select control bit (RTS or DTR)

(7) Will handshaking be used? If not, will RTS-CTS loopback be needed to fool the DCE?

(8) Is the NBS-10 at the end of the network?

(9) Is the NBS-10 being configured the only RS-485 card on the network? Are there any other bias resistors on the network?

(10) At what baud rate should the card run? At what speed do the other devices on the network operate?

2.4 SETTING THE BASE ADDRESS

The NBS-10 card occupies eight consecutive locations in the PC I/O address space. Dip Switch 1 is used to set the base address for these locations. Since existing ports in your PC occupy some I/O addresses, it is important that you find an unoccupied address area to put your NBS-10. The following table shows an I/O map representing an amalgamation of PC, XT and AT peripheral locations. This sort of list is very difficult to maintain due to the rapid advance of PC peripheral products.

SHADED AREA FOR SYSTEM BOARD USE ONLY

HEX RANGE DECODED	# OF BYTES	FUNCTION
000H TO 001FH	32	DMA CHIP
0020H TO 003FH	32	INTERRUPT CONTROLLER
0040H TO 005FH	32	TIMER COUNTER
0060H TO 007FH	32	PPI
0080H TO 009FH	32	DMA PAGE REGISTERS
00A0H TO 00BFH	32	NMI MASK BIT
00C0H TO 01FFH	320	NOT USED
0200H TO 020FH	16	GAME CONTROL ADAPTER
0210H TO 0277H	104	NOT USED
0278H TO 027FH	8	LPT2: PRINTER PORT
0280H TO 02DFH	96	NOT USED
02E0H TO 02E7H	8	VGA/EGA/GPIB/DATA ACQ.
02E8H TO 02EFH	8	COM4: SERIAL PORT - NOT STANDARD
02F0H TO 02F7H	8	NOT USED
02F8H TO 02FFH	8	COM2: SECOND SERIAL PORT
0300H TO 031FH	32	PROTOTYPE CARD
0320H TO 0324H	4	FIXED DISK REGISTERS
+0325H TO 0347H	34	NOT USED
0348H TO 0357H	10	DCA 3278
0358H TO 035FH	8	NOT USED
0360H TO 036FH	16	RESERVED
0370H TO 0371H	2	NOT USED
0372H TO 0377H	6	DISKETTE CONTROLLER
0378H TO 037FH	8	LPT1: PRINTER PORT
0380H TO 038FH	8	SDLC, BISYNCHRONOUS 2
0390H TO 0393H	4	CLUSTER
0394H TO 039FH	12	NOT USED
03A0H TO 03AFH	16	BISYNCHRONOUS 1
03B0H TO 03BFH	16	MONOCHROME + PRINTER CARD
03C0H TO 03CFH	16	VIDEO REGISTERS
03D0H TO 03DFH	16	CGA ADAPTER
03E0H TO 03E7H	8	NOT USED
03E8H TO 03EFH	8	COM3: SERIAL PORT
03F0H TO 03F7H	8	5.25" DISK DRIVE
03F8H TO 03FFH	8	COM1: SERIAL PORT

This table is not exhaustive. Further information can be found in the following books: System BIOS by Phoenix, Upgrading and Repairing your PCs by Mueller, and Interfacing to the IBM Personal Computer by Eggbrecht. See the reference list (section 1.4) for more information on these books.

Place your NBS-10 in an address that is not used by any other adapter in your system. As this list is a composite, chances are that you will not have many of the peripherals mentioned above. If you want to use DOS or BIOS serial port services, we recommend that you put the card in one of the COM ports.

The following table shows address locations and corresponding switch settings for COM1 through COM4. In addition, the last line of the table shows the correspondence between switch number and PC address line.

COM PORT	**BASE ADDRESS**	**SW 1.1**	**SW 1.2**	**SW 1.3**	**SW 1.4**	**SW 1.5**	**SW 1.6**	**SW 1.7**
COM1	03F8 HEX	OFF	OFF	OFF	OFF	OFF	OFF	OFF
COM2	02F8 HEX	OFF	OFF	OFF	OFF	OFF	ON	OFF
COM3*	03E8 HEX	OFF	ON	OFF	OFF	OFF	OFF	OFF
COM4*	02E8 HEX	OFF	ON	OFF	OFF	OFF	ON	OFF
	ADDRESS LINE	A3	A4	A5	A6	A7	A8	A9

* NON-STANDARD

NOTE: A switch in the ON or CLOSED position corresponds to a 0 in the address whereas a switch in the OFF or OPEN position corresponds to a 1 in the address. For example if the NBS-10 were to be placed at address 210H, we would start by converting 210H to binary (210H = 1000010000). Since the last three binary digits will be used to select the registers on the UART, we will ignore them (210H = 1000010XXX).

Switch 1.1 corresponds to address line A3 (the least significant bit). Translate the binary address starting from the right and proceeding left. Where you see a 1 in the binary representation, set the switch to open or off and where you see a 0, set the switch to closed or on. The appropriate switch settings for address 210H are shown in the table below:

ADDRESS LINE	A0	A1	A2	A3	A4	A5	A6	A7	A8	A9
210H	X	X	X	0	1	0	0	0	0	1
SWITCH NUMBER	X	X	X	SW 1.1	SW 1.2	SW 1.3	SW 1.4	SW 1.5	SW 1.6	SW 1.7
SWITCH SETTING	X	X	X	ON	OFF	ON	ON	ON	ON	OFF

2.5 INTERRUPT REQUEST SELECT

After setting the boards base address, the next step to tackle is setting the interrupt request. If you are using the NBS-10 in a COM port address, the following table can help you pick the proper interrupt.

COM PORT	BASE ADDRESS	INTERRUPT REQUEST
COM1	03F8 HEX	IRQ4
COM2	02F8 HEX	IRQ3
COM3	03E8 HEX	IRQ4
COM4	02E8 HEX	IRQ3

If you are not using one of the COM ports, check your software for information on which interrupt to use. If interrupt driven transfers are not required, remove the jumper completely.

2.6 OUTPUT CONTROL SWITCH SETTINGS

The NBS-10 flexible output circuits control the following features: Duplex, Receiver/Transmitter enablers and functions, and CTS-RTS loopback.

OUTPUT CONTROL SWITCH SETTING	
SWITCH	FUNCTION
SW 2.1	ON = HALF DUPLEX NOTE: the setting of SW 2.1 affects SW 2.2 OFF = FULL DUPLEX
SW 2.2	in HALF SUPLEX MODE: ON = RECEIVER ALWAYS ON OFF = RECEIVER OFF DURING TRANSMIT; controled by RTS or DTR (see SW 2.4) in FULL DUPLEX MODE: ON = TRANSMITTER ENABLED BY RTS or DTR (see SW 2.4) OFF = TRANSMITTER ALWAYS ON
SW 2.3	ON = RTS-CTS LOOPBACK ENABLED OFF = RTS-CTS LOOPBACK DISABLED
SW 2.4	ON = TRANSMIT (RECEIVE) WHEN DTR = 1 OFF = TRANSMIT (RECEIVE) WHEN RTS = 1
SW 2.5	*** NOT CURRENTLY DEFINED ***

2.6.1 EXPLANATION OF TABLE: Switch Functions

2.6.1.1 SWITCH 2.1: DUPLEX CONTROL

Definitions

HALF DUPLEX: transmission of data in either direction on the same conductor pair, but not simultaneous transmission in both directions.

FULL DUPLEX: simultaneous, two-way transmission on two independent conductor pairs--one pair for each direction.

When Switch 2.1 is in the off position, the NBS-10 is in full duplex mode. One pair of wires should be connected to the transmit data connector terminals on the NBS-10 board and another pair of wires should be attached to the NBS-10 receive data connector terminals. Full duplex is used in most RS-422 configurations and in four-wire RS-485 systems. (If a four-wire RS-485 system is used, a control line must be selected to control the enabling and disabling of the transmitter so that other nodes can utilize the lines. Please refer to the description of switches 2.4 further on in this section for information on this control line.)

When Switch 2.1 is in the on position, the NBS-10 is in half duplex mode. In half duplex mode, there is one and only one pair of conductors. This conductor pair is attached to the transmit data connector which is internally connected on the NBS-10 board to a receive pair. Half duplex is used in most RS-485 applications including embedded control networking. In half duplex mode, the NBS-10 card must be told whether it is receiving or transmitting data; this information is provided by user software. The transmitter/receiver direction is governed by the user's software protocol, which allows other nodes to share the line. Please refer to the description of switch 2.4 further on in this section for more information on the selection of this control signal.

2.6.1.2 SWITCH 2.2 RECEIVE-TRANSMIT CONFIGURATION

The function of this switch is determined by the position of switch 2.1.

If SW2.1 is ON (in half duplex mode), SW2.2 (on or off) controls the receiver. When SW2.2 is ON, the receiver is always enabled. This feature should be utilized on networks where collision detection is part of the communications protocol. Every time a character is transmitted, it appears in the receive buffer of the UART. If the data transmitted does not equal the data received, a collision or other error has occurred. If SW2.2 is OFF, then the receiver is disabled whenever the transmitter is enabled. Please refer to the description of SW2.4 further on in this section for more information.

If SW2.1 is OFF (in full duplex mode), SW2.2 (on or off) controls the transmitter. If SW2.2 is OFF, then the transmitter is always on. This is the normal condition for RS-422 communications. If the SW2.2 is ON, then the transmitter is enabled by RTS or DTR. This configuration is useful in four-wire full duplex RS-485 systems in which there is a multidrop pair for each direction of transmission. For example, an RS-485 master-slave network with slave interrupt capabilities would utilize this configuration.

2.6.1.3 SWITCH 2.3 RTS-CTS LOOPBACK

This feature is used when a software protocol in either the transmitting device or the receiving device requires handshaking signals that the other unit does not supply. The SW2.3 allows a "request to send" handshaking signal to be looped back to the "clear to send" signal, in essence fooling the handshaking dependent device into believing that the appropriate handshake has been delivered. This switch saves the user from having to make an external loopback on the connector.

2.6.1.4 SWITCH 2.4 CONTROL SELECT

Switch 2.4 allows the selection of the control signal used to drive the features selected by SW2.2. RTS and DTR are available control sources; this selection allows for compatibility with other manufacturers' products.

2.6.1.5 SWITCH 2.5 IS CURRENTLY UNDEFINED

2.7 USING DOS SERVICES

While possible, but not recommended, the NBS-10 can be used with software that employs DOS serial port service routines. Good software packages do not use DOS serial port services because they are slow and not interrupt driven. Before performing the desired serial port action, DOS asserts the DTR (Data Terminal Ready) signal and then waits for an active (true) response on DCD (Data Carrier Detect) and DSR (Data Set Ready) lines. If these lines are not true, the serial port will not function properly.

When DOS services are desired, all interface signals must be implemented for the serial port to function properly. Connecting a device that omits all handshaking signals to a device that expects them results in an interface that does not work. In order to use DOS services in the above condition, DOS must be tricked into thinking that the proper connections exist.

Because the NBS-10 does not provide DTR, DCD and DSR connections, DTR is internally looped back to DCD and DSR on the card. DOS services should work with the NBS-10 without any assistance from the user.

2.8 TERMINATION RESISTORS AND TRANSMISSION LINE BIAS

2.8.1 TERMINATION RESISTORS ON RS-485 NETWORKS

In an RS-485 network configuration only two termination resistors are permitted. These resistors must be located at opposites ends of the physical line. When an NBS-10 resides at the physical end of the network, termination resistors J2, J3 and J4 need to be installed (Refer to Figure 2.3). If not at the physical end of the line, the jumpers should be removed. The NBS-10 cards should be attached to the transmission line with short stub wires or daisy chained directly together.

NBS-10 AT THE END OF THE NETWORK: INSTALL JUMPERS J2,J3,J4
NBS-10 NOT AT THE END OF THE NETWORK: REMOVE JUMPERS J2,J3,J4

2.8.2 BIAS ON RS-485 NETWORKS

If the NBS-10 is serving as the network master for an RS-485 network and no other bias networks are provided, the DIP resistor package R5 should be installed. If there are other bias resistors on the network, or if RS-422 is being used, then the resistor pack should be removed. For a more complete discussion of bias resistor theory, the reader is referred to section 2.1.

2.8.3 TERMINATION RESISTORS ON RS-422 NETWORKS

If the NBS-10 is used on an RS-422 network, one termination resistor must exist at the end of the transmission line opposite the transmitter. If the NBS-10 is the transmitter on the network, then the J2 termination resistors should removed. If the NBS-10 is the last receiver on the network, termination resistors J3 and J4 should be installed. Only if the NBS-10 is the last receiver on the network should the termination resistors be installed.

2.8.4 BIAS ON RS-422 NETWORKS

Since a transmitter always holds the line in a known state, bias resistors are not required.

2.9 DEFAULT FACTORY SWITCH SETTINGS

ADDRESS SETTING							
SWITCH #	SW 1.1	SW 1.2	SW 1.3	SW 1.4	SW 1.5	SW 1.6	SW 1.7
2F8H	1	1	1	1	1	0	1
SETTING	OFF	OFF	OFF	OFF	OFF	ON	OFF

INTERRUPT = IRQ3

OUTPUT CONTROL SETTINGS					
SWITCH #	SW 2.1	SW 2.2	SW 2.3	SW 2.4	SW 2.5
SETTING	OFF	OFF	OFF	OFF	OFF
PURPOSE	FULL-DUPLEX	TX ALWAYS ON	LOOP BACK OFF	RTS CTRL SOURCE	NOT DEFINED

BIAS RESISTORS R5 INSTALLED

ALL TERMINATION RESISTORS INSTALLED: J2, J3, J4

18.432 MHz CLOCK OSCILLATOR

2.10 EXTERNAL CONNECTOR PINOUTS

J6 AND J7 RS-485 D-9 CONNECTOR

Connectors J6 and J7 are shorted together on the NBS-10. Both genders of connector are provided for convenience only.

D-9 PIN#	FUNCTION	COMMENT
1	RTS+ (REQUEST TO SEND +)	OPTIONAL HANDSHAKE OUTPUT
6	RTS- (REQUEST TO SEND -)	OPTIONAL HANDSHAKE OUTPUT
2	TXD+ (TRANSMIT DATA +)	ALSO HALF DUPLEX PAIR
7	TXD- (TRANSMIT DATA -)	ALSO HALF DUPLEX PAIR
3*	NETWORK COMMON +	TIED TO PC GROUND THRU 100 OHMS
4	RXD+ (RECEIVE DATA +)	NOT USED IN HALF DUPLEX
8	RXD- (RECEIVE DATA -)	NOT USED IN HALF DUPLEX
5	CTS+ (CLEAR TO SEND +)	OPTIONAL HANDSHAKE INPUT
9	CTS- (CLEAR TO SEND -)	OPTIONAL HANDSHAKE INPUT

*the NBS-10 provides a 100 ohm resistor between Pin 3 (the network common) and the PC ground. This resistor limits ground currents. For more information on grounding, see RS-485 and local and national electric codes.

SHIELDED CABLE MUST BE USED IN ORDER TO COMPLY WITH FCC REGULATIONS.

2.11 INSTALLING THE NBS-10 IN THE HOST COMPUTER

After you have completed configuration of the NBS-10, you are ready for installation. The NBS-10 occupies one slot in most ISA PC compatible computers. Follow the manucturers disassembly instructions on your particular machine in order to remove the cover. Locate a free slot in which to put your NBS-10 and remove the cover. Locate a free slot in which to put your NBS-10 and remove the blank I/O plate. Please note: slot eight on the original IBM PC XT and Portable computer are not regular expansion ports and should not be used. Insert the NBS-10 into the slot making sure that the edge-card and the bus connector mate firmly. Replace the bracket screw and then replace the cover.

If you are making your own cables, be sure to use shielded twisted pair wire and shielded metal head shells. The NBS-10 has been verified for compliance with FCC Part 15 regulations using shielded cable and head shells. These components help protect against unwanted radio frequency emissons emanating from your computers circuitry and they also protect your system from external electromagnetic interference.

3 PROGRAMMING THE NBS-10

3.1 INTRODUCTION

The NBS-10 uses Intel's 82510 Asynchronous Serial Controller (a fancy UART), which includes many high-performance features and allows emulation of the 16450 UART (used in the IBM PC/AT serial port). The 82510 is functionally identical to the 16450 on powerup, so you should be able to use your existing programs with little or no modification. High-performance features include four-byte FIFOs, baud rates up to 288,000 baud (using a 18.432 MHz crystal oscillator), an extra timer, and nine-bit communication modes.
Intel has kindly allowed us to reprint their 82510 data sheets, which are located in Appendix B of this manual. Rather than replicate the information they contain, this chapter will focus on using this chip from a programmer's perspective. We urge you to consult other references on serial-port programming as well, such as Intel's AP-401 and the chapter on the serial port in Duncan [7] (8250/16450 programming).

3.2 FOR SERIOUS APPLICATIONS

We have come up with the following recommendations for programming the NBS-10, which are especially relevant if you are running at moderate or high baud rates. Feel free to ignore these recommendations if you are running at low baud rates and do not care if you lose characters once in a while.

1. Manipulate the NBS-10 by directly writing to and reading from its registers. Don't use BIOS or MS-DOS services, or commands in high-level languages which use those services; they are slow and not particularly flexible, and they do not use the high-performance features of the 82510 UART.

2. Write a serial port interrupt handler which buffers incoming and outgoing data in assembly language. Compared to polling, using an interrupt handler means that the 82510 will be serviced faster, allowing higher baud rates to be sustained without losing incoming characters. Duncan [7] contains a good sample program.

3. Use the FIFOs in the NBS-10's 82510, especially if you are polling the NBS-10 instead of following recommendation #2. If for some reason your program can not promptly read a character from the serial port, the receiver FIFO will buffer up to 4 characters, reducing the probability that you will lose any. In addition, the FIFOs can reduce the amount of CPU time required to service the 82510.

3.3 THE BASICS

The 82510 includes 35 registers grouped into four banks:

bank 0: 8250A/16450-compatible bank
bank 1: general work bank
bank 2: general configuration
bank 3: modem configuration

If you only use 16450 emulation mode, you can work entirely in bank 0. However, if you use the high-performance features of the 82510, you will typically configure the NBS-10 using banks 0, 2, and 3, then operate in bank 1. In order to switch register banks, you write to the GIR/BANK register, which is present in all four banks.

Regardless of what language you use to program the NBS-10, you will have to set the frame format and mode of operation. Frame format consists of three parts: the number of data bits per frame, the number of stop bits per frame, and the type of parity. In addition to the 16450 emulation mode frame formats (five to eight data bits, with or without parity), the 82510 is capable of nine-data-bit formats.

When a nine-data-bit format is chosen, the 82510 can operate in a mode, in which the only messages which cause a CPU interrupt from the NBS-10 are messages with a particular destination address. This reduces the amount of time the CPU must expend doing network communication.

Next, you have to decide how to send and receive data. There are two techniques for doing this; polling and interrupt-driven transfer. Polling requires the program to periodically check the status of the UART and sending or receiving characters as appropriate. To do interrupt-driven transfer, the NBS-10 must be programmed to generate an interrupt request when the UART requires service, and an interrupt service routine must be written. Polling is simpler to program, but interrupt-driven transfer allows higher baud rates and reduces the probability of lost characters because the UART can be serviced sooner.

You can use the standard high-level language serial port commands to program the NBS-10 (although we do not recommend it), but those commands only support 16450 emulation mode. Low-level commands which allow direct access to the 82510's registers allow full use of the NBS-10's capabilities. Here are some useful commands from BASIC and C (see the reference manual for your particular compiler or interpreter):

BASIC:
OPEN COM, GET, PUT, and LOC are useful high-level commands. In some versions of BASIC, OPEN COM causes serial data to be buffered. INP and OUT are low-level commands which can directly access 82510 registers.

C:
fopen() is a high-level function which uses DOS services. bios_serialcom() (Microsoft C) and bioscom() (Turbo C) are medium-level functions which use BIOS services. inp() and outp() are low-level functions which can directly access 82510 registers.

If you program in assembly language, you can use BIOS interrupt 14H (see below), DOS interrupt 21H, or directly program the 82510's registers using the 8086-family IN and OUT instructions.

Those of you who are considering programming the NBS-10 using low-level commands may wish to refer to the sample C-language program at the end of this chapter. Although it uses some procedures which are unique to nine-bit mode programming, much of it is applicable to standard serial-port programming. As previously mentioned, Duncan [7] contains a good sample program written in assembly language which includes a serial-port interrupt handler for the 8250/16450.

3.4 USE OF BIOS AND DOS SERVICES

Both IBM PC BIOS and MS-DOS provide serial communication services which should work fine with the NBS-10. However, you will probably be limited to using the 82510 in 16450 emulation mode. Of the two, BIOS services are more flexible. But both are slow, and neither are interrupt-driven, so programs written at Cimetrics Technology generally do not use them. See Duncan [7] for detailed information about these services.

The IBM PC BIOS provides serial communication services through software interrupt 14H. These include:

- initialize COM port
- send char
- read char
- get COM port status

MS-DOS includes device drivers for COM ports which should work with the NBS-10. The device drivers can be accessed using DOS commands such as MODE and COPY, as well as software interrupt 21H.

3.5 SETTING THE BAUD RATE

The NBS-10 is shipped with an 18.432 Mhz crystal oscillator, allowing the NBS-10 to run at up to 288,000 baud. However, in 16450 emulation mode, the NBS-10 behaves as if it were a 16450 with a 1.8432 Mhz crystal oscillator like a standard PC/XT/AT, with a speed limit of 57,600 baud. This means that you may be able to use high-level language commands or BIOS services or DOS services to set the baud rate if you intend to run in 16450 emulation mode. Otherwise you will have to directly program the 82510's registers.

The baud rate is controlled by one or both of the 82510's 16-bit timers. Each timer has two eight-bit registers which combine to form a sixteen-bit divisor when the timer is used as a baud-rate generator. In 16450 emulation mode, the two timers are cascaded, and the divisor (written to timer A) is calculated using the following formula:

```
divisor = (oscillator frequency in Hz) / (160 * baud rate)
```

If you use only one timer, then the divisor is calculated using the following formula:

```
divisor = (oscillator frequency in Hz) / (32 * baud rate)
```

To use only timer A for baud-rate generation, you will have to write values to registers in the modem configuration bank:

```
RegisterBank(3);
/* put timer B in timer mode */
outp(BRGB_CONFIGURATION_ADDRESS,0);
/* set baud-rate clock source to timer A: */
outp(CLOCKS_CONFIGURE_ADDRESS,0x50);
```

In the following example, the divisor value is written to timer A's two registers:

```
unsigned char lcrValue, divisorHighByte, divisorLowByte; unsigned int divisor;
...
divisorHighByte = (unsigned char)(divisor >>>> 8);  /* usually, this is 0 */
divisorLowByte = (unsigned char)(divisor & 0x00FF);

RegisterBank(0);
lcrValue = inp(LINE_CONTROL_ADDRESS); /* save LCR value */
outp(LINE_CONTROL_ADDRESS, 0x80);     /* set DLAB */
outp(DIVISOR_A_LOW_BYTE_ADDRESS, divisorLowByte);
outp(DIVISOR_A_HIGH_BYTE_ADDRESS, divisorHighByte);
outp(LINE_CONTROL_ADDRESS, lcrValue); /* restore prev. LCR value */ ...
```

Here are the divisor values for some common baud rates:

(18.432 MHz oscillator, BRG-A only)

Baud Rate	Divisor	Comments
187500	3	(12 MHz 8096)
83333	7	(16 MHz 8051)
62500	9	(12 MHz 8051)
56000	10	(11.059 MHz 8051)
19200	30	
9600	60	
1200	480	(high byte = 1, low byte = 224)

3.6 NINE-DATA-BIT MODES

As mentioned previously, the 82510's frame formats include a nine-data-bit format. This format is useful when the NBS-10 is used to allow a PC to be in a network of embedded controllers, most often as the master node of a polled master-slave network. Many popular eight- and sixteen-bit embedded controllers have a nine-bit mode, including the 8051, 68HC11, Z180, and the 8096.

Best use of the built-in capabilities of certain microcontrollers means that a protocol with a certain structure must be used. In this type of protocol, a message between two network nodes consists of several nine-bit words: an address word followed by data words. The two kinds of words have the following format (MSB-LSB):

1XXXXXXXX address word
0XXXXXXXX data word

The address word specifies the receiving node. Data words can contain commands and command parameters, node status information, check words,and of course, data.

When this kind of protocol is used, you may choose to use the 82510's µLAN mode, which is useful when you want to minimize CPU interruptions. In this mode, the 82510 will only accept messages with specific destination addresses.

Entering nine-data-bit mode is simple:

```
RegisterBank(0);
outp(LINE_CONTROL_ADDRESS,0);
RegisterBank(2);
outp(TRANSMIT_MACHINE_MODE_ADDRESS,0x20);
```

To send a character, set the ninth bit, then write the lower eight bits to the transmit data register:

```
/* RegisterBank(1); */
outp(TRANSMIT_FLAGS_ADDRESS, 0x20);  /* set ninth bit */
/* to clear the ninth bit: outp(TRANSMIT_FLAGS_ADDRESS,0); */
while ((inp(FIFO_LEVEL_ADDRESS) & 0x07) == 4) ;
       /* wait until there is room in the transmitter FIFO */
outp(TRANSMIT_DATA_ADDRESS, lowerEightBits);
```

To receive a character, read the ninth bit, then read the lower eight bits from the receive data register:

```
/* RegisterBank(1); */
ninthBit = inp(RECEIVE_FLAGS_ADDRESS) & 1;
lowerEightBits = inp(RECEIVE_BUFFER_ADDRESS);
```

3.7 USING FIFO MODE

We strongly recommend that you make use of the 82510's FIFO mode for the following reasons:

1. With standard UARTs, the CPU must read a character received by the UART before the next character has arrived or else a character will be lost; this puts an upper limit on the baud rate. Using an interrupt handler will allow higher baud rates than polling, but in some circumstances the handling of the interrupt may be preempted by a higher-priority interrupt. Since the 82510's receiver FIFO can buffer up to four characters, the probability of losing incoming characters is lower.

2. The FIFOs reduce the amount of time that the CPU must use in servicing the UART. For example, the receive interrupt trigger level can be set so that a data available interrupt does not occur until several characters have been received; therefore, the interrupt routine would be executed fewer times. Likewise, up to four bytes to be transmitted can be transferred to the 82510 at a time.

If you are using an interrupt handler or are using nine-bit mode, you will need to decide on the receive interrupt trigger threshold. This is the minimum number of characters in the receiver FIFO required to trigger a data available interrupt, and can be one to four bytes. This threshold can be varied while the UART is running to improve performance. The higher the threshold, the less time the CPU spends servicing the UART, but higher thresholds increase the probability of lost characters.

If the receive interrupt trigger threshold is greater than 1, then it is possible for data to remain in the receiver FIFO for an extended period of time without the 82510 interrupting the CPU. There are two ways to handle this problem: keep the receive interrupt trigger threshold at 1 byte, or use Timer B to periodically interrupt the CPU (see Intel AP-401 for an example of the latter).

Entering FIFO mode is simple:

```
RegisterBank(2);
outp(INTERNAL_MODE_ADDRESS,0x08); /* RX FIFO depth=4 bytes */
outp(FIFO_MODE_ADDRESS,0);
/* RX FIFO interrupt if 1 or more bytes are in the RX FIFO */
/* TX FIFO interrupt if the TX FIFO is empty */
```

Once the NBS-10 is initialized, you can determine the number of characters in each FIFO by reading the FIFO level register.

3.8 NBS-10 INTERRUPTS AND THE PC

As discussed in the 82510 data sheets, the 82510 asynchronous serial controller can generate an interrupt for any one of several different reasons. These interrupts can be enabled by setting appropriate bits in the general enable register and optionally other registers. However, the NBS-10 is designed so that a UART interrupt will not generate an interrupt request on the IBM PC or AT bus unless bit OUT2 (modem control register) is also set. In addition, the PC's 8259 interrupt controller must be appropriately programmed.

Because of the design of the PC and AT computers, multiple peripherals can not simultaneously use the same IRQ line. However, it is possible to have two peripherals share an IRQ line as long as they are not using the line at the same time. For example, you could put both a modem and an NBS-10 on IRQ 3, but you could not do interrupt-driven modem communication at the same time you were doing interrupt-driven NBS-10 communication.

APPENDIX A

SAMPLE PROGRAM

The following C-language program uses an extremely simple nine-bit protocol to allow two or more NBS-10 equipped PC's to "talk" to each other over a single shielded twisted pair cable. Reception is done by polling the Receive FIFO Interrupt Request (RFIR) bit in the General Status Register (GSR [register #20]) to see if any characters have been received. If there are characters in the FIFO, then the Receive Ninth Data Bit (RND) in the Receive Flags Register (RXF [register #24]) is checked to see if an address byte is present. If the ninth bit is set, then the address byte is compared with the address of this node. If the message is intended for this node, characters are put into a ring buffer. When time permits, received characters are removed from the ring buffer and displayed (putchar() function).

APPENDIX A: SAMPLE PROGRAM

```
/*
 *      NBS-10 talk program:
 *      Send messages between PCs on a two-wire RS-485 network using
 *      an extremely simple nine-bit protocol.
 *
 *      SW1 settings: (for COM2, address 2F8h: see BASE_ADDRESS)
 *              1-5, 7 ON
 *              6 OFF
 *
 *       SW2 settings:
 *               1 ON   (half-duplex)
 *              2 OFF   (receiver off during transmit)
 *              3 OFF   (RTS-CTS loopback disabled)
 *              4 OFF
 *              5 OFF   (RTS=1 enables transmit: see TX_ENABLE_BIT)
 *
 *      Interprocessor message format:
 *              one address character (ninth-bit set)
 *              Data characters (ninth-bit clear)
 */

char *copyright = "Copyright (c) 1991 by Cimetrics, Inc.  All rights reserved.";
char *version = "*** 4/1/91 ***";

#include <stdio.h>
#include <conio.h>
#include <string.h>
#include <math.h>

#define BASE_ADDRESS 0x2F8    /* COM2 = 2F8 */
#define TX_ENABLE_BIT RTS_BIT /* RTS_BIT or DTR_BIT */

/*** type definitions ***/
typedef unsigned char uchar;
typedef int boolean;

/*** 82510 register addresses ***/
        /* bank 0 */
#define EMULATE_16450_BANK 0
#define DIVISOR_LOW_ADDRESS (BASE_ADDRESS+0)
#define DIVISOR_HIGH_ADDRESS (BASE_ADDRESS+1)
#define TRANSMIT_BUFFER_ADDRESS (BASE_ADDRESS+0)     /* also bank 1 */
#define RECEIVE_BUFFER_ADDRESS (BASE_ADDRESS+0)              /* also bank 1 */
#define INTERRUPT_ENABLE_ADDRESS (BASE_ADDRESS+1)
#define INTERRUPT_ID_ADDRESS (BASE_ADDRESS+2)        /* any bank */
#define BANK_ADDRESS (BASE_ADDRESS+2)         /* any bank */
#define LINE_CONTROL_ADDRESS (BASE_ADDRESS+3)
#define MODEM_CONTROL_ADDRESS (BASE_ADDRESS+4)       /* also bank 1 */
#define LINE_STATUS_ADDRESS (BASE_ADDRESS+5)
#define MODEM_STATUS_ADDRESS (BASE_ADDRESS+6)
#define SCRATCH_ADDRESS (BASE_ADDRESS+7)
        /* bank 1 */
#define GENERAL_WORK_BANK 1
#define TRANSMIT_FLAGS_ADDRESS (BASE_ADDRESS+1)
#define TRANSMIT_FLAGS_BANK 1
#define RECEIVE_FLAGS_ADDRESS (BASE_ADDRESS+1)
#define RECEIVE_FLAGS_BANK 1
#define GENERAL_STATUS_ADDRESS (BASE_ADDRESS+7)
#define GENERAL_STATUS_BANK 1
#define FIFO_LEVEL_ADDRESS (BASE_ADDRESS+4)
#define FIFO_LEVEL_BANK 1
#define INTERNAL_COMMAND_ADDRESS (BASE_ADDRESS+7)
#define INTERNAL_COMMAND_BANK 1
        /* bank 2 */
#define TRANSMIT_MACHINE_MODE_ADDRESS (BASE_ADDRESS+3)
#define TRANSMIT_MACHINE_MODE_BANK 2
#define INTERNAL_MODE_ADDRESS (BASE_ADDRESS+4)
#define INTERNAL_MODE_BANK 2
#define FIFO_MODE_ADDRESS (BASE_ADDRESS+1)
#define FIFO_MODE_BANK 2
        /* bank 3 */
#define CLOCKS_CONFIGURE_ADDRESS (BASE_ADDRESS+0)
#define CLOCKS_CONFIGURE_BANK 3
#define BRGB_CONFIGURATION_ADDRESS (BASE_ADDRESS+3)
#define BRGB_CONFIGURATION_BANK 3
```

```
/*** useful UART register bits ***/
#define RTS_BIT 0x02          /* modem control register */
#define DTR_BIT 0x01
#define TxIR_BIT 0x10         /* general status register */
#define RFIR_BIT 0x01
#define RESET_BIT 0x10        /* internal command register */

/*** allocate ring buffer ***/
#define RING_BUFFER_SIZE 100
char ringBuffer[RING_BUFFER_SIZE];
int ringInPointer = 0;
int ringOutPointer = 0;

/*** allocate other buffers ***/
char lowerEightBuffer[1000];
char scratchBuffer[100];

/*** miscellaneous constants and variables ***/
#define ESC_KEY 27
#define FALSE 0
#define TRUE (!FALSE)
#define MAX_XTAL_FREQ 20.0
#define MIN_XTAL_FREQ 1.0
double baudRate = 9600.0;            /* default, in bits per second (baud) */
double crystalFrequency = 18.432;    /* default, in MHz */
unsigned int thisNodeAddress;

#define RegisterBank(b) outp(BANK_ADDRESS,(b)<<5);

/*** function prototypes: ***/
void main(void);
void NetworkMonitor(void);
int ReceiveChar(int *ninthBit);
void SendMessage(uchar nodeAddress, unsigned int messageLength);
void TransmitMode(void);
void ReceiveMode(void);
void SetNinthBit(void);
void ClearNinthBit(void);
void SendChar(uchar lowerEightBits);
void EnableFifoMode(void);
void SetBaudRate(double rate);
unsigned short ComputeDivisor(double baudRate, boolean suppressErrorMsg);


/*** This is the first procedure executed when the program is run. ***/
void main(void)
{
       puts("NBS-10 talk program, for 2-wire (half-duplex) RS-485.");
       puts(copyright);
       puts(version);
       RegisterBank(INTERNAL_COMMAND_BANK);
       outp(INTERNAL_COMMAND_ADDRESS,RESET_BIT);    /* reset 82510 */
       RegisterBank(EMULATE_16450_BANK);

              /*** SETUP UART REGISTERS ***/
       do {
              printf("Crystal Frequency in MHz? [%.4lf MHz] ",crystalFrequency);
              gets(scratchBuffer);
              sscanf(scratchBuffer,"%lf",&crystalFrequency);
       } while (crystalFrequency < MIN_XTAL_FREQ && crystalFrequency > MAX_XTAL_FREQ);
       do {
              printf("Speed in baud? [%.1lf baud] ",baudRate);
              gets(scratchBuffer);
              sscanf(scratchBuffer,"%lf",&baudRate);
       } while (ComputeDivisor(baudRate,FALSE) == 0);

       SetBaudRate(baudRate);
       outp(MODEM_CONTROL_ADDRESS, 0);
              /* receive mode, disable IRQ, disable loopback */
       /* nine-bit mode: */
       RegisterBank(EMULATE_16450_BANK);
       outp(LINE_CONTROL_ADDRESS,0);
       RegisterBank(TRANSMIT_MACHINE_MODE_BANK);
       outp(TRANSMIT_MACHINE_MODE_ADDRESS,0x20);
       EnableFifoMode();
       outp(INTERRUPT_ENABLE_ADDRESS,0);     /* disable all interrupts */
       inp(LINE_STATUS_ADDRESS);     /* clear out any garbage */
       inp(RECEIVE_BUFFER_ADDRESS);  /* ditto */

       /*** OPERATOR TYPES IN ADDRESS NUMBER FOR THIS NODE ***/
```

```
        thisNodeAddress = 32767;
        while (thisNodeAddress > 255) {
                printf("What address # is this node? (0 - 255) ");
                gets(scratchBuffer);
                sscanf(scratchBuffer,"%u",&thisNodeAddress);
        }

        NetworkMonitor();

                /*** CLEANUP UART REGISTERS BEFORE TERMINATING ***/
        RegisterBank(INTERNAL_COMMAND_BANK);
        outp(INTERNAL_COMMAND_ADDRESS,RESET_BIT);    /* reset 82510 */
}

void NetworkMonitor(void)
{
        static boolean messageBeingReceived = FALSE;
        int ninthBit, c;
        unsigned int destinationNodeAddress = 0;

        RegisterBank(GENERAL_WORK_BANK);
        printf("Network monitor running: press Esc to exit, any other key to send message.\n");
        /*** EXECUTE THIS LOOP UNTIL THE OPERATOR PRESSES THE ESCAPE KEY ***/
        for (;;) {
                /*** GET ALL DATA RECEIVED BY THE UART ***/
                while ((inp(GENERAL_STATUS_ADDRESS) & RFIR_BIT)) {
                        c = ReceiveChar(&ninthBit);
                        if (ninthBit == 1) {   /* address char */
                                /* this character begins a new message */
                                if ((unsigned int)c == thisNodeAddress) {
                                        /* message is addressed to me */
                                        messageBeingReceived = TRUE;
                                } else {
                                        /* message is not addressed to me */
                                        messageBeingReceived = FALSE;
                                }
                        } else if (messageBeingReceived) {
                                /*** PUT DATA FOR ME IN THE RING BUFFER ***/
                                if (++ringInPointer >= RING_BUFFER_SIZE)
                                        ringInPointer = 0;
                                ringBuffer[ringInPointer] = (char)c;
                        } /* otherwise ignore the character */
                }
                /*** DISPLAY A CHARACTER IN THE RING BUFFER, IF ANY ***/
                if (ringInPointer != ringOutPointer) {
                        if (++ringOutPointer >= RING_BUFFER_SIZE)
                                ringOutPointer = 0;
                        putchar(ringBuffer[ringOutPointer]);
                                /* do something with the char */
                }
                /*** CHECK TO SEE IF THE OPERATOR HAS PRESSED A KEY ***/
                if (kbhit()) {
                        if (getch() == ESC_KEY) {
                                RegisterBank(EMULATE_16450_BANK);
                                return; /* Exit */
                        }
                        printf("\nSend message to which node? [%u] (0 - 255) ",
                                destinationNodeAddress);
                        /* OPERATOR TYPES ADDRESS OF DESTINATION NODE */
                        gets(scratchBuffer);
                        if (sscanf(scratchBuffer,"%u",&destinationNodeAddress)) {
                                printf("Message? ");
                                /* OPERATOR TYPES MESSAGE */
                                gets(scratchBuffer);
                                strcpy(lowerEightBuffer,scratchBuffer);
                                /*** TRANSMIT THE MESSAGE ***/
                                SendMessage((uchar)destinationNodeAddress,
                                            strlen(lowerEightBuffer));
                        }
                }
        }
}

int ReceiveChar(int *ninthBit)
/*** RETURNS THE LOWER EIGHT BITS, AND SETS VARIABLE ninthBit ***/
{
        /* assumes GENERAL_WORK_BANK */
        *ninthBit = inp(RECEIVE_FLAGS_ADDRESS) & 1;
        return inp(RECEIVE_BUFFER_ADDRESS);
}
```

```
void SendMessage(uchar nodeAddress, unsigned int messageLength)
/* message was previously put into lowerEightBuffer[] */
{
        unsigned int i;

        TransmitMode();
        SetNinthBit();
        SendChar(nodeAddress); /* ADDRESS CHAR: 9TH BIT IS SET */
        ClearNinthBit();
        for (i=0; i<messageLength;) {  /* DATA CHARACTERS: 9TH BIT IS CLEAR */
                SendChar(lowerEightBuffer[i++]);
        }
        ReceiveMode();
}

void TransmitMode(void)
/*** ENABLE RS-485 DRIVERS ***/
{
        /* assumes EMULATE_16450_BANK or GENERAL_WORK_BANK */
        outp(MODEM_CONTROL_ADDRESS, TX_ENABLE_BIT);
                /* set transmit enable bit */
}

void ReceiveMode(void)
/*** AFTER TRANSMISSION IS COMPLETE, DISABLE RS-485 DRIVERS ***/
{
        /* assumes GENERAL_WORK_BANK */
        while ((inp(GENERAL_STATUS_ADDRESS) & TxIR_BIT) == 0) ;
                /* wait until transmission is complete */
        outp(MODEM_CONTROL_ADDRESS, 0);
                /* clear transmit enable bit, whatever it is */
}

void SetNinthBit(void) /* for transmission */
{
        /* assumes GENERAL_WORK_BANK */
        outp(TRANSMIT_FLAGS_ADDRESS, 0x20);
}

void ClearNinthBit(void) /* for transmission */
{
        /* assumes GENERAL_WORK_BANK */
        outp(TRANSMIT_FLAGS_ADDRESS, 0);
}

void SendChar(uchar lowerEightBits)
/* SET OR CLEAR NINTH BIT BEFORE CALLING THIS PROCEDURE */
{
        /* assumes GENERAL_WORK_BANK */
        while ((inp(FIFO_LEVEL_ADDRESS) & 0x07) == 4) ;
                /* wait until there is room in the transmitter FIFO */
        outp(TRANSMIT_BUFFER_ADDRESS, lowerEightBits);
}

void EnableFifoMode(void)
{
        RegisterBank(INTERNAL_MODE_BANK);
        outp(INTERNAL_MODE_ADDRESS,0x08);     /* RX FIFO depth = 4 bytes */
        RegisterBank(FIFO_MODE_BANK);
        outp(FIFO_MODE_ADDRESS,0);    /* RX FIFO threshold = 1 byte */
        RegisterBank(EMULATE_16450_BANK);
        printf("82510 FIFO mode enabled; trigger at 1 byte\n\r");
}

void SetBaudRate(double rate)
/* rate is in baud; maximum and minimum depend on the crystal frequency */
{
        unsigned short divisor;
        uchar lcrValue;

        divisor = ComputeDivisor(rate, TRUE);
        if (divisor != 0) {
                /* disable second divisor: */
                RegisterBank(BRGB_CONFIGURATION_BANK);
                outp(BRGB_CONFIGURATION_ADDRESS,0);
                /* set clock source to first divisor: */
                RegisterBank(CLOCKS_CONFIGURE_BANK);
                outp(CLOCKS_CONFIGURE_ADDRESS,0x50);
                RegisterBank(EMULATE_16450_BANK);
```

```
                /* set value of first divisor */
                lcrValue = (uchar)inp(LINE_CONTROL_ADDRESS);
                outp(LINE_CONTROL_ADDRESS,lcrValue | 0x80);     /* DLAB on */
                outp(DIVISOR_LOW_ADDRESS,(uchar)(divisor & 0x00FF));
                outp(DIVISOR_HIGH_ADDRESS,(uchar)((divisor>>8) & 0x00FF));
                outp(LINE_CONTROL_ADDRESS,lcrValue);    /* DLAB restored */
        }
}

unsigned short ComputeDivisor(double baudRate, boolean suppressErrorMsg)
{
        double rawDivisor, divisor, error;

        rawDivisor = crystalFrequency / (2 * 16e-6 * baudRate);
                /* factor of 2 is because we are using an external oscillator */
        divisor = floor(rawDivisor + .5);     /* round to nearest integer */
        if (!suppressErrorMsg) {
                if (divisor > 65535.0) {
                        printf("A baud rate of %.1lf is too low.\n",baudRate);
                        return 0;
                }
                if (divisor < 1.0) {
                        printf("A baud rate of %.1lf is too high.\n",baudRate);
                        return 0;
                }
                error = fabs((divisor/rawDivisor) - 1.0);
                if (error > 0.02)
                        printf("\aWARNING: the baud rate generation error will be %.2lf%%.\n",
                                100.0 * error);
        }
        return (unsigned short)divisor;
}
```