CIMETRICS TECHNOLOGY

NBS-2 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-2 HARDWARE

Cimetrics Technology warrants the NBS-2 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

TABLE OF CONTENTS

1 OVERVIEW

1.1 DESCRIPTION OF THE NBS-2

The NBS-2 is a flexible RS-530/485/422 general purpose asynchronous serial adapter for PC/XT/AT compatible computers. Based on the National Semiconductor NS16550AF UART, the NBS-2 provides several features particularly suited to RS-485 multidrop networking. The NBS-2 can also be used as an RS-422 serial link using the EIA-530 (TYPE SR) connector standard.

FEATURES
- Half duplex 9-bit serial communication mode
- Half- or full-duplex standard PC communications
- Flexible control of transmit direction - RTS/DTR/OUT1
- 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-2 features and gives an overview of this manual.

CHAPTER 2: INSTALLATION
Provided detailed instructions and flowcharts on how to set up the NBS-2 for operation.

CHAPTER 3: PROGRAMMING
Describes the operation of the NBS-2 in typical programming applications.

CHAPTER 4: NINE-BIT MODE
Describes programming of the NBS-2 in its nine-bit serial communication mode.

APPENDIX A:
Sample Program

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] Data Communications, Local Area Networks, UARTs Handbook (1990), National Semiconductor Corp., pp. 4-36 to 4-56 (NS16550AF section) and pp. 4-57 to 4-83 (AN-491 section).

[10] Embedded Control Applications, Intel Corp., 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.

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 and television communications. This equipment has been tested and found to comply with the limits for a Class B computing device pursuant to Subpart B, Part 15 of the FCC Rules, which are designed to provide reasonable protection against such interference in a residential area.

However, there is no guarantee that interference to radio or television reception will not occur in a particular installation. If this equipment does cause interference, the user at his own expense will be required to take whatever measures may be required to correct the interference.

The Federal Communications Commission has a publication entitled *Interference Handbook*, which may be helpful to you. This booklet (stock number 004-000-004505-7) may be purchased from the Superintendent of Documents, U.S. Government Printing Office, Washington, D.C. 20402

The NBS-2 has been verified for FCC Part 15 Class B 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-2. 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-530/485/422, 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-485 and RS-422

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. RS-530 is the EIA standard which describes the **MECHANICAL** interface and pin-out that is recommended for RS-422 applications.

(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.)

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 2.2, 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]

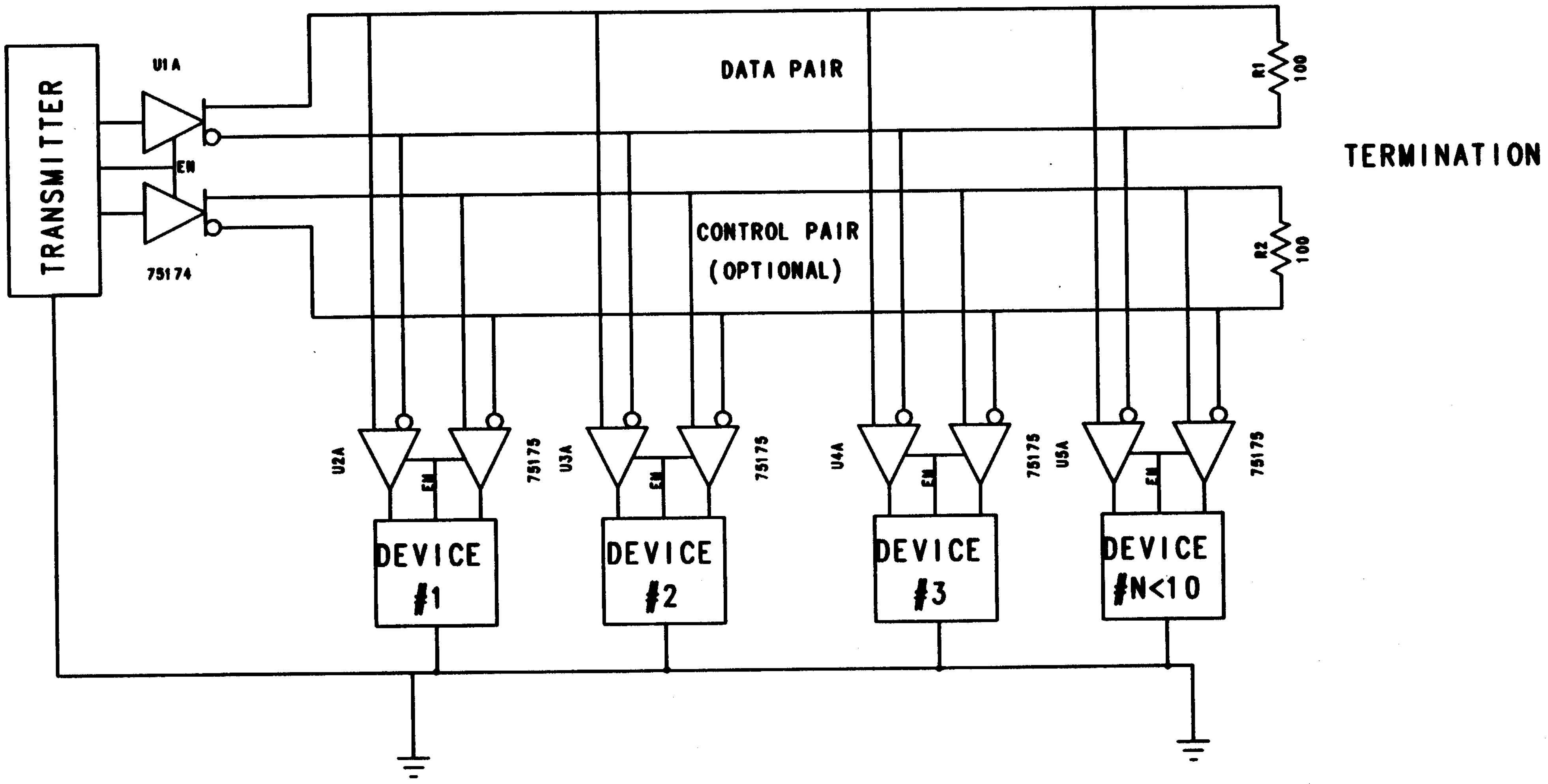
COMPARISON OF RS-232, RS-422, RS-485

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

2.2.2 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 (100Ω in the NBS-2) in order to limit current.

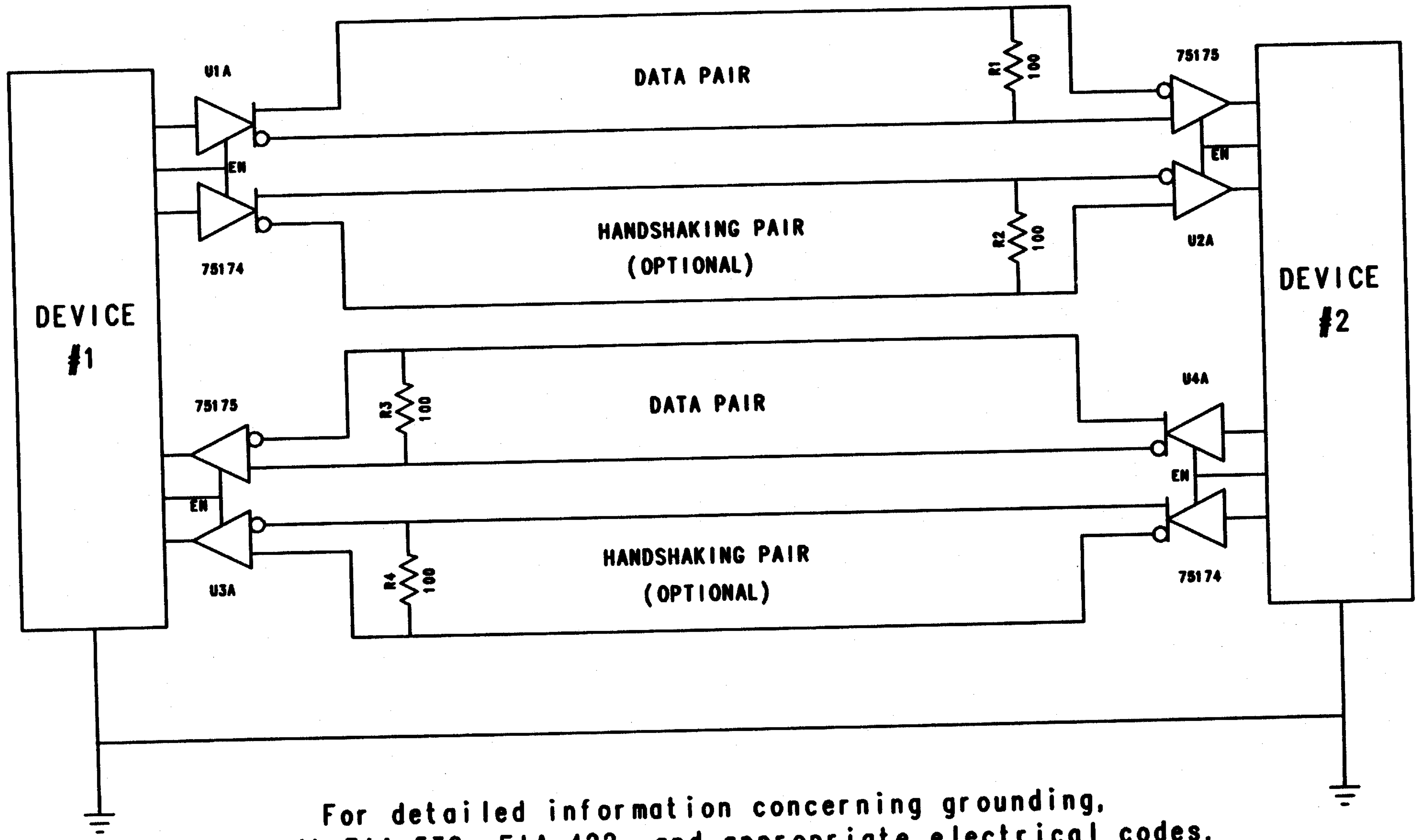
RS-422 MULTIPLE RECEIVER CONFIGURATION



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

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.

2.2.3 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 wreaks 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:

(1) Additional bias networks affect the impedance of the transmission line;

(2) 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-2 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.

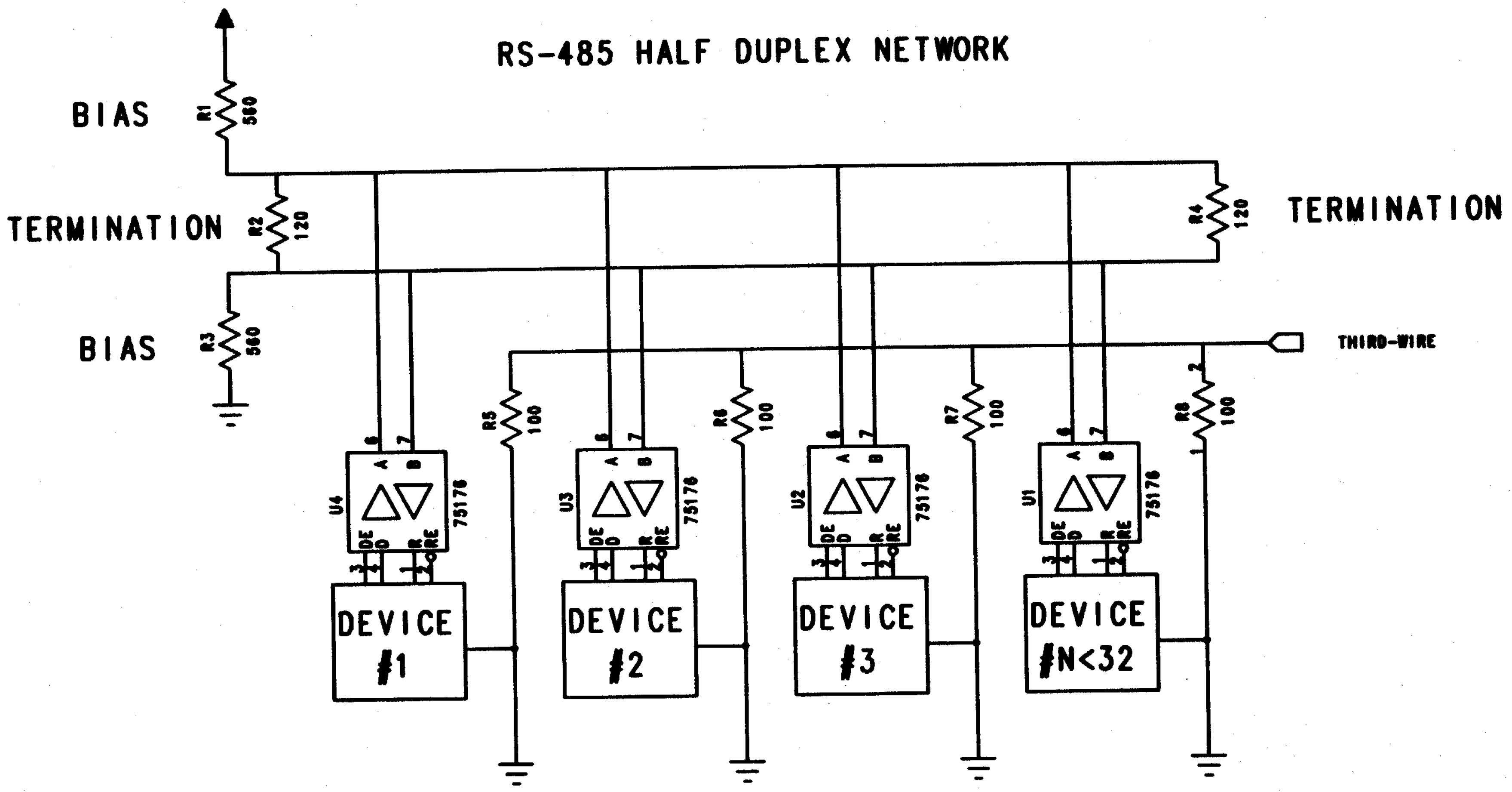
Termination resistors should be placed only at the ends of the transmission line. There should be ONLY TWO SUCH RESISTORS ON THE ENTIRE NETWORK. More than two will substantially lower the impedance of the line and will hinder the line drivers.

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

2.2.4 RS-530 (EIA-530)

RS-530 is effectively a specification for a connector and pinout to be used with RS-422. As we mentioned earlier, the RS-422 and RS-485 include only electrical specifications. RS-530 was released in 1987; in the years before it was available, the lack of connector pinout gave rise to a host of non-standard interfaces. NBS-2 utilizes the RS-530, which specifies asynchronous modem control signals, synchronous clock signals, test and loopback signals. This standard encompasses various configurations; since the NBS-2 is an asynchronous device, it has implemented an asynchronous type SR (send-receive) interface. (Refer to Section 2.10 for pinout).

RS-485 HALF DUPLEX NETWORK



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

FIGURE 2.3

2.3 NBS-2 CONFIGURATION WORKSHEET

Before you get started on the NBS-2 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-2? The NBS-2 requires eight consecutive locations above the base address.
(2) What interrupt will the NBS-2 use?

(3) For what application will the NBS-2 be used?
    (A) RS-530/422 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,DTR,OUT1)
    (B) Should receiver be off during transmit?

(6) If full duplex:
    (A) Should the transmitter always be on?
    (B) If no, select control bit (RTS,DTR,OUT1)

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

(8) Is the NBS-2 at the end of the network?

(9) Is the NBS-2 being configured as 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?

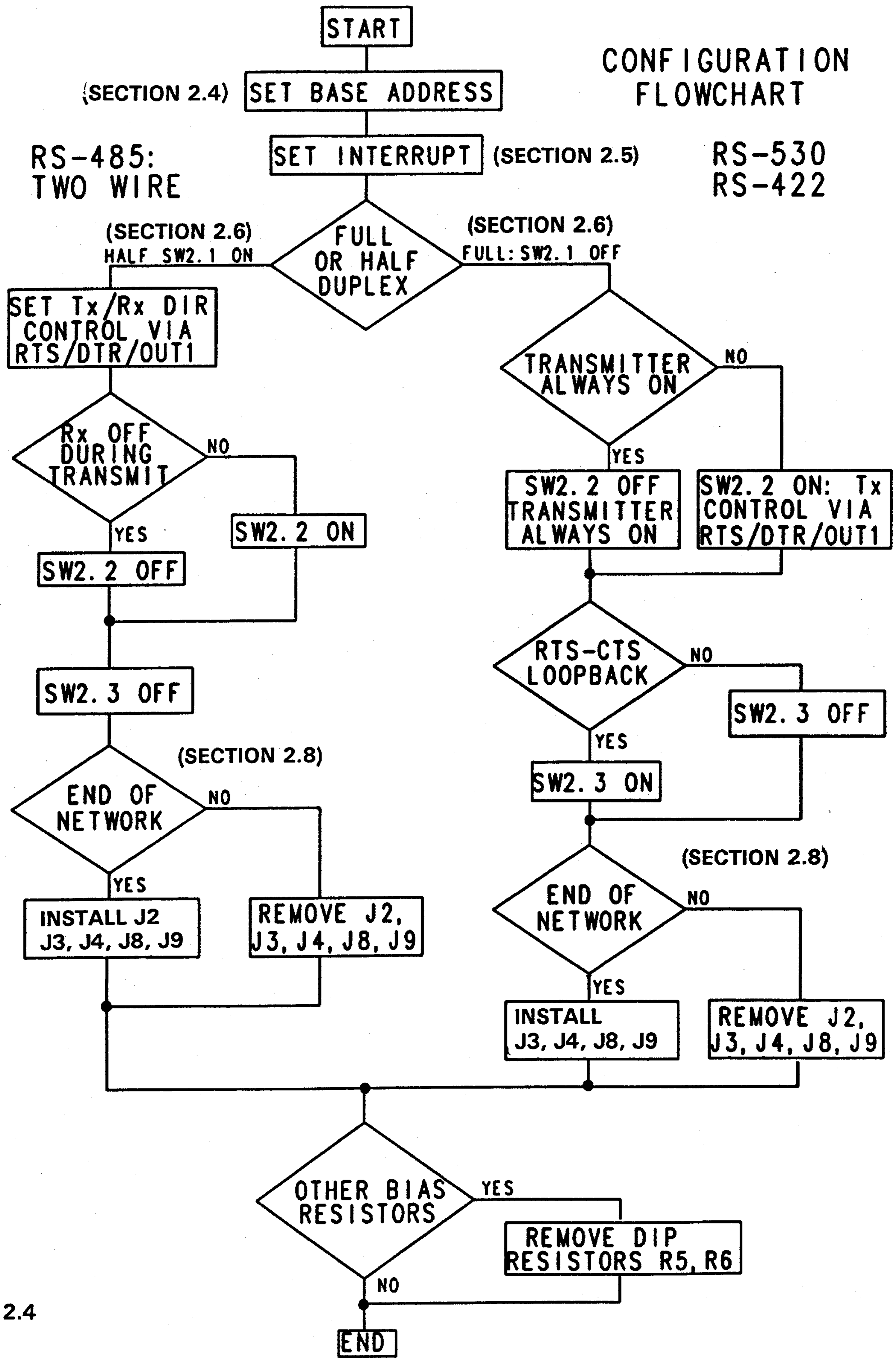

START
(SECTION 2.4)
SET BASE ADDRESS
CONFIGURATION FLOWCHART
RS-485: TWO WIRE
SET INTERRUPT
(SECTION 2.5)
RS-530 RS-422
(SECTION 2.6)
HALF SW2.1 ON
FULL OR HALF DUPLEX
(SECTION 2.6)
FULL: SW2.1 OFF
SET Tx/Rx DIR CONTROL VIA RTS/DTR/OUT1
TRANSMITTER ALWAYS ON
NO
Rx OFF DURING TRANSMIT
NO
YES
SW2.2 OFF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

FIGURE 2.4

2.4 SETTING THE BASE ADDRESS

The NBS-2 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-2. 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 Pheonix, 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-2 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 recomend that you put the card in one of the COM ports.

The following table shows address locations and corresponding switch locations 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

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. So for example if the NBS-2 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 swtich to closed or on. The appropriate swtich 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-2 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, DTR, or OUT1 (see SW 2.4 and SW 2.5)
	in FULL DUPLEX MODE:
	ON = TRANSMITTER ENABLED BY RTS, DTR, or OUT1 (see SW 2.4 and SW 2.5)
	OFF = TRANSMITTER ALWAYS ON	

SW 2.3	ON = RTS-CTS LOOPBACK ENABLED
	OFF = RTS-CTS LOOPBACK DISABLED	

SW 2.4	SW 2.5	
OFF	OFF	TRANSMIT (RECEIVE) WHEN RTS = 1(0)
ON	ON	TRANSMIT (RECEIVE) WHEN DTR = 1(0)
OFF	ON	TRANSMIT (RECEIVE) WHEN OUT1 = 1(0)
ON	OFF	*** 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-2 is in full duplex mode. One pair of wires should be connected to the transmit data connector terminals on the NBS-2 board and another pair of wires should be attached to the NBS-2 receive data connector terminals. Full duplex is used in most RS-530/422 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 and 2.5 further on in this section for information on this control line.)

When Switch 2.1 is in the on position, the NBS-2 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-2 board to a receive pair. Half duplex is used in most RS-485 applications including embedded control networking. In half duplex mode, the NBS-2 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 and 2.5 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 and 2.5 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-530/422 communications. If the SW2.2 is ON, then the transmitter is enabled by RTS/OUT1/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 AND 2.5 CONTROL SELECT

Switches 2.4 and 2.5 allow the selection of the control signal used to drive the features selected by SW2.2. RTS, DTR and OUT1 are available control sources; this selection allows for compatibility with other manufacturers' products.

2.7 CONFIGURING FOR DOS SERVICES

While possible, but not recommended, the NBS-2 can be configured for software that uses 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.

If you are not supplying these signals, but still want to use the DOS serial port services, make the following connections in your D-25 cable headshell. Pins 6, 8 and 20 need to be connected together. Also pins 10, 22 and 23 need to be connected together. Regardless of the connector type (D-9 or D-25) you are using, this loopback connector must be made. These connections will allow the serial port to function properly.

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-2 resides at the physical end of the network, termination resistors J2, J3, J4, J8 and J9 need to be installed (Refer to Figure 2.4 in Section 2.2). If not at the pyhsical end of the line, the jumpers should be removed. The NBS-2 cards should be attached to the transmission line with short stub wires or daisy chained directly together.

NBS-2 AT THE END OF THE NETWORK: INSTALL JUMPERS J2,J3,J4,J8,J9
NBS-2 NOT AT THE END OF THE NETWORK: REMOVE JUMPERS J2,J3,J4,J8,J9

2.8.2 BIAS ON RS-485 NETWORKS

If the NBS-2 is serving as the network master for an RS-485 network and no other bias networks are provided, the the DIP resistor packages R5 and R6 should be installed. If there are other bias resistors on the network, or if RS-422 is being used, then the resistor packs should be removed. For a more complete discussion of bias resistor theory, the reader is refered to section 2.1.

2.8.3 TERMINATION RESISTORS ON RS-422 NETWORKS

If the NBS-2 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-2 is the transmitter on the network, then the J2 termination resistors should removed. If the NBS-2 is the last receiver on the network, termination resistors J3 and J4 should be installed. Only if the NBS-2 is the last receiver on the network should the termination resistors be installed. (Refer to Figure 2.1).

2.8.4 BIAS ON RS-422 NETWORKS

Since a transmitter always holds the line in a known state, bias resistors are not required. Remove R5 and R6 for this configuration.

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 AND R6 INSTALLED

ALL TERMINATION RESISTORS INSTALLED: J2, J3, J4, J8, J9

8 MHz CLOCK OSCILLATOR

NBS-2 BOARD LAYOUT AND JUMPER POSITIONS

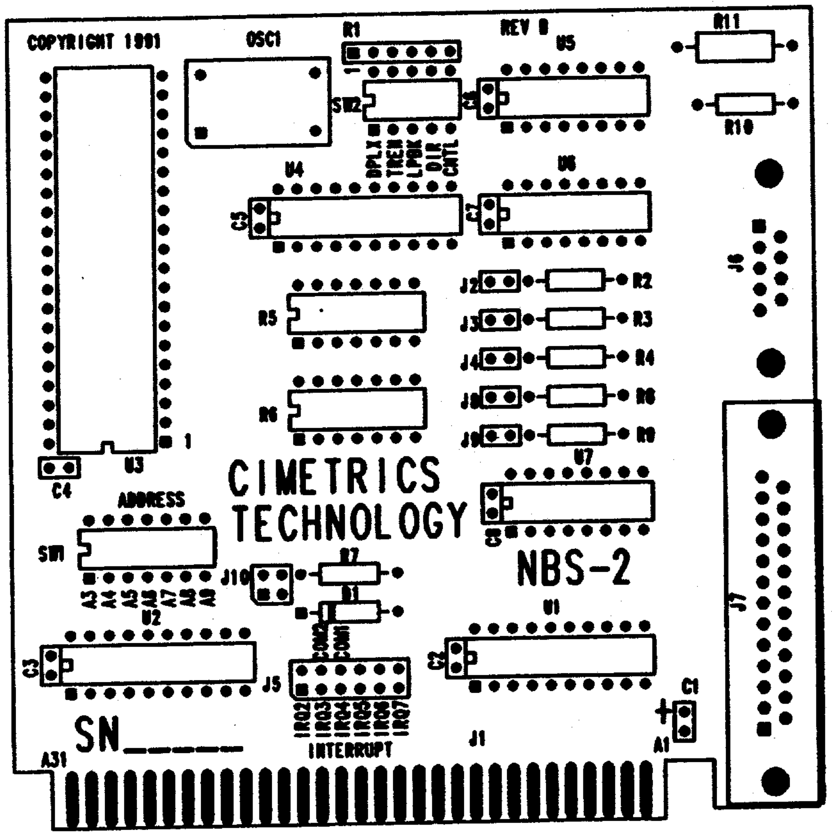


FIGURE 2.5

2.10 EXTERNAL CONNECTOR PINOUTS

J7 RS-530/422 D-25 CONNECTOR				
D-25 PIN#	RS-530 SIGNAL	RS-530 CIRCUIT	CIRCUIT TYPE (I/O)	FUNCTION
1	SHIELD		SHEILD	CONNECT ONLY AT DTE END
2	TD +	BA (A)	DATA (OUT)	TRANSMIT DATA +
14	TD -	BA (B)	DATA (OUT)	TRANSMIT DATA -
3	RD +	BB (A)	DATA (IN)	RECEIVE DATA +
16	RD -	BB (B)	DATA (IN)	RECEIVE DATA -
4	RTS +	CA (A)	CONTROL (OUT)	REQUEST TO SEND +
19	RTS -	CA (B)	CONTROL (OUT)	REQUEST TO SEND -
5	CTS +	CB (A)	CONTROL (IN)	CLEAR TO SEND +
13	CTS -	CB (B)	CONTROL (IN)	CLEAR TO SEND -
6	DCE + (DSR)	CC (A)	CONTROL (IN)	DCE READY + (DATA SET READY)
22	DCE - (DSR)	CC (B)	CONTROL (IN)	DCE READY - (DATA SET READY)
20	DTE + (DTR)	CD (A)	CONTROL (OUT)	DTE READY + (DATA TERMINAL READY)
23	DTE - (DTR)	CD (B)	CONTROL (OUT)	DTE READY - (DATA TERMINAL READY)
7	SIGNAL GND	AB	GROUND	SIGNAL GROUND (PC GROUND) ***
8	RLSD + (DCD)	CF (A)	CONTROL (IN)	RECEIVED LINE SIGNAL DETECTOR +
10	RLSD - (DCD)	CF (B)	CONTROL (IN)	RECEIVED LINE SIGNAL DETECTOR -
15*	TSET +	DB (A)	TIMING*	TX SIGNAL ELEMENT TIMING + (DCE)
12*	TSET -	DB (B)	TIMING*	TX SIGNAL ELEMENT TIMING - (DCE)
17*	RSET +	DD (A)	TIMING*	RX SIGNAL ELEMENT TIMING + (DCE)
9*	RSET -	DD (B)	TIMING*	RX SIGNAL ELEMENT TIMING - (DCE)
18*	LOCAL LPBK	LL	CONTROL*	LOCAL LOOPBACK
21*	REMOTE LPBK	RL	CONTROL*	REMOTE LOOPBACK
24*	TSET +	DA (A)	TIMING*	TS SIGNAL ELEMENT TIMING + (DTE)
11*	TSET -	DA (B)	TIMING*	TS SIGNAL ELEMENT TIMING - (DTE)
25	TEST MODE	TM	CONTROL (IN)	TEST MODE: CONDDECTED TO RING IND.

[...] COMMON RS-232 TYPE DESCRIPTION WHEN DIFFERENT THAN RS-530
{...} SIGNAL SOURCE

* NOT SUPPORTED BY THE NBS-2: The NBS-2 implements the TYPE SR (send-receive) data transmission configuration specified by the EIA-530 standard. Because the NBS-2 is an asynchronous device, the timing signals have no relevance. The remote and local loopback control functions are optional in the Type SR interface and since the UART does not directly support either of these functions, they have been omitted.

*** The RS-530/422 CONNECTOR PROVIDES A DIRECT CONNECTION TO THE PC GROUND. THERE IS NO 100 OHM CURRENT LIMITING RESISTOR BETWEEN PIN 7 AND GROUND.

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

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-2 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-2 IN THE HOST COMPUTER

After you have completed configuration of the NBS-2, you are ready for installation. The NBS-2 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-2 and remove the cover. Locate a free slot in which to put your NBS-2 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-2 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-2 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-2 AS A STANDARD SERIAL CARD

3.1 INTRODUCTION

The NBS-2 uses National Semiconductor's NS16550AF UART, which is very similar to the 16450 UART used in the IBM PC/AT serial port. In fact, the 16550 is functionally identical to the 16450 on powerup, so you should be able to use your existing programs with very little modification. However, the 16550 also includes receiver and transmitter FIFOs, which when used can reduce the amount of time your PC spends servicing the NBS-2, as well as lessen the probability that you will lose characters at high baud rates.

National Semiconductor has kindly allowed us to reprint their NS16550AF data sheets, which are located in the Appendix 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 the chapter on the serial port in ***Duncan*** [7].

3.2 FOR SERIOUS APPLICATIONS

We have come up with the following recommendations for programming the NBS-2, 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-2 by directly writing to and reading from its registers. Don't fool around with BIOS or MS-DOS services, or commands in high-level languages which use them; they are slow and not particularly flexible.

2. Write a serial port interrupt handler which buffers incoming and outgoing data in assembly language. Compared to polling, use of an interrupt handler means that the UART will be serviced faster, allowing higher baud rates to be sustained without losing incoming characters. ***Duncan***[7] contains a good sample program which taught us a lot about this subject.

3. Use the FIFOs in the NBS-2's NS16550AF UART, especially if you are polling the NBS-2 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 16 characters, greatly reducing the probability that you will lose any. In addition, the FIFOs can reduce the amount of CPU time required to service the 16550.

3.3 THE BASICS

Regardless of what language you use to program the NBS-2, you will have to set several parameters of operation: the baud rate and the frame format. 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.

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-2 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-2 (although we do not recommend it), but you will probably have to use low-level commands to set the baud rate (see "Setting the Baud Rate", below). 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 16550 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 16550 registers.

If you program in assembly language, you can use BIOS interrupt 14H (see below), DOS interrupt 21H, or directly program the 16550 registers using the 8086-family IN and OUT instructions. Those of you who are considering programming the NBS-2 using low-level commands may wish to refer to the sample C-language program in the nine-bit mode programming chapter. Although it uses techniques 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.

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-2 except for setting the baud rate (see "Setting the Baud Rate", below). We recommend ***Duncan***[7] for detailed information about these services. 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.

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 fine with the NBS-2, except for setting the baud rate (see "Setting the Baud Rate", below). The device drivers can be accessed using DOS commands such as MODE and COPY, as well as software interrupt 21H. However, you may have to deal with the handshaking lines for correct operation. For example, we found that in order to execute the DOS command "COPY SOMEFILE.TXT COM1:", we had to connect the DTR, DSR, and DCD lines on the transmitting NBS-2 (see section 2.8 for more information).

3.5 SETTING THE BAUD RATE

The NBS-2 is shipped with an 8 MHz crystal oscillator, allowing the NBS-2 to run at up to 250,000 baud. However, the standard PC/XT/AT serial port uses a 1.8432 MHz crystal or oscillator, and has an upper speed limit of 57,600 baud. This means that if you use high-level language commands or BIOS services or DOS services to set the baud rate on an NBS-2, the actual baud rate will be 4.34 times as high as what you asked for, which is generally not a useful baud rate! Commercial serial communication software will also most likely assume that the crystal frequency is 1.8432 MHz, with the same undesirable result. There are two ways around this problem. The first is to directly program the 16550's two divisor latches using the low-level commands (e.g. inp() and outp()) after you have invoked the high-level serial-port initialization commands, as described below. The second solution (much less desirable) is to order a 1.8432 MHz crystal oscillator from Cimetrics Technology, imposing an upper speed limit of 57,600 baud; those of you with commercial serial communication software to run will most likely choose this solution.

You have probably figured out by now that the baud rate is determined by the crystal oscillator frequency (up to 8 MHz in the 16550) and the 16550's two divisor latches. The divisor latches are eight-bit registers which combine to form a sixteen-bit divisor. The divisor is calculated using the following formula:

```
>divisor = (oscillator frequency in Hz) / (16 * baud rate)
```

That divisor value is written to the two divisor latches:

In C:

```
>unsigned char lcrValue, divisorHighByte, divisorLowByte;
>unsigned int divisor;
...
>divisorHighByte = (unsigned char)(divisor >> 8);  /*
>usually, this is 0 */
divisorLowByte = (unsigned char)(divisor & 0x00FF);
lcrValue = inp(LINE_CONTROL_ADDRESS); /* save LCR value */
outp(LINE_CONTROL_ADDRESS, 0x80);     /* set  DLAB */
outp(DIVISOR_LOW_BYTE_ADDRESS, divisorLowByte);
outp(DIVISOR_HIGH_BYTE_ADDRESS, divisorHighByte);
outp(LINE_CONTROL_ADDRESS, lcrValue); /* restore prev. LCR
value */       ...
```

Here are the divisor values for some common baud rates:
(8 MHz oscillator)

Baud Rate	Divisor	
25600	2	
83333	6	
62500	8	
56000	9	
19200	26	
9600	52	
1200	417	(high byte = 1, low byte = 224)

3.6 USING FIFO MODE

We strongly recommend that you make use of the NS16550AF'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 16550's receiver FIFO can buffer up to 16 characters, the probability of losing incoming characters is much 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 16 bytes to be transmitted can be transferred to the UART 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 1, 4, 8, or 14 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.

Sometimes, the amount of data in the receiver FIFO remains below the interrupt trigger threshold for an extended period of time. If this condition persists, then the 16550 will trigger a timeout interrupt.

3.7 NBS-2 INTERRUPTS AND THE PC

As discussed in the NS16550AF data sheets, the 16550 UART can generate an interrupt for any one of five different reasons. These interrupts can be enabled by setting appropriate bits in the interrupt enable register. However, the NBS-2 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 both using the line at the same time. For example, you could put both a modem and an NBS-2 on IRQ-3, but you could not do interrupt-driven modem communication at the same time you were doing interrupt-driven NBS-2 communication.

Even if you decide not to do interrupt-driven NBS-2 communication, you may still wish to make use of the 16550's interrupt capability. If you enable certain 16550 interrupts but do not set the OUT2 bit, then you can poll the interrupt identification register. This polling technique is especially useful in nine-bit mode, as described in the nine-bit mode programming chapter.

4 9-BIT SERIAL COMMUNICATION

4.1 INTRODUCTION

The NBS-2 has a half-duplex nine-data-bit serial asynchronous communication mode. Programming the NBS-2 in this mode is quite similar to standard serial card programming; however, certain procedures must be followed for correct operation. This chapter describes those programming procedures, and includes programming examples in C.

We recommend that you read the chapter entitled "Programming the NBS-2 as a Standard Serial Card" before reading this one. The NBS-2 uses the NS16550AF UART; if you are not very familiar with this chip, you may wish to refer to the data sheets in the Appendix of this manual, as we will be working with the 16550's registers throughout this chapter.

4.2 WHEN SHOULD NINE-BIT MODE BE CONSIDERED?

The NBS-2's nine-data-bit serial communication mode was designed for use in low-cost half-duplex RS-485 networks which include 8051, 8096, 68HC05, 68HC11, or Z180 family microcontrollers. These microcontrollers have a nine-data-bit asyncronous serial communication mode which is particularly appropriate for networking. However, the NBS-2's nine-data-bit mode may be appropriate for networks which do not include those microcontrollers.

4.3 THE NINE-BIT MODE

The NBS-2's nine-bit mode has one start bit, nine data bits starting with the least significant bit, and finally one or two stop bits. There is no parity bit.

The UARTs found in most PC serial cards (8250, 16450, or 16550), including the NBS-2, were not really designed for nine-data-bit communication. However, we can make use of the 16550's eight-data-bits plus stick-parity mode to do nine-bit communication; we call this technique Parity Modulation. During transmission, the ninth bit is controlled by putting the NBS-2 into either even stick parity mode (0) or odd

stick parity mode (1). During reception, the NBS-2 is put into even stick parity mode, and the received ninth bit is the Line Status Register's parity error bit. With a 16550 UART, Parity Modulation only works for half-duplex communication.

The IBM-PC BIOS and MS-DOS were not designed for nine-data-bit serial communication. Thus, when you are in nine-bit mode, you should not use BIOS serial communications functions (interrupt 14h) or MS-DOS functions to control the NBS-2, or high-level language statements which make use of BIOS or DOS services. If you are programming the NBS-2 using a high-level language such as BASIC or C, you should only use the very-low-level I/O port input and output functions:

BASIC:

use INP and OUT
do not use OPEN COM, GET, PUT, LOC, etc.

C:

use inp() and outp()
in Microsoft C, do not use _bios_serialcom()
in Turbo C, do not use bioscom()

4.4 RECEIVING NINE-BIT DATA

Data reception in nine-bit communication mode is done using the 16550's even stick parity mode, which is set by writing to the line control register.

We recommend that your programs use an interrupt routine to handle received data, especially if you are running at moderate or high baud rates. However, it is possible to handle character reception by polling. If you decide to do polling, do not poll on the data ready bit of the line status register; instead, enable the data ready interrupt in the interrupt enable register (but do not set OUT2) and poll the interrupt identification register until a received data available interrupt or a character timeout interrupt is indicated.

The NBS-2's NS16550AF UART has a receive FIFO. We strongly recommend that you use it (see "Programming the NBS-2 as a Standard Serial Card"), as it will reduce the probability that you will lose characters and it can reduce the amount of UART service time.

Receive procedure:

0. Set UART to eight-data-bit, even-stick-parity mode (line control register). Enable and clear the FIFOs (FIFO control register). Enable the received data available interrupt (interrupt enable register). If you are using an interrupt routine, set bit OUT2 (modem control register).

1. Read the interrupt ID register; when a received data available interrupt or a character timeout indication interrupt occurs, proceed to step 2.

2. Read the line status register. The parity error bit is the ninth data bit.

3. Read the receive buffer register, which contains the lower eight data bits.

```
int ReceiveChar(int *ninthBit)
{
       *ninthBit = ((inp(LINE_STATUS_ADDRESS) & PE_BIT) ? 1 : 0);  return
inp(RECEIVE_BUFFER_ADDRESS);
}
```

4. Goto step 1.

4.5 SENDING NINE-BIT DATA

Data transmission in nine-bit communication mode is done using the 16550's stick parity modes, which are set by writing to the line control register.

Here are the steps required to send nine-bit data:

1. Set the transmit enable bit (modem control register); this will be either RTS, DTR, or OUT1, depending on your NBS-2 switch settings.

```
void TransmitMode(void)
{
       outp(MODEM_CONTROL_ADDRESS, TX_ENABLE_BIT);
              /* set transmit enable bit */
}
```

2. Do steps 3 and 4 for each character.

3. If necessary, change the ninth bit by changing the parity bit (line control register). But before changing the parity bit, make sure that bit TEMT is set (line status register).

ninth bit = 1: odd stick parity
ninth bit = 0: even stick parity

```
void SetNinthBit(void) /* for transmission */
{
       while ((inp(LINE_STATUS_ADDRESS) & TEMT_BIT) == 0) ;
          /* wait until previous characters have been transmitted */
outp(LINE_CONTROL_ADDRESS, 0x2B); /* odd stick parity */ }

 void ClearNinthBit(void)
{
       while ((inp(LINE_STATUS_ADDRESS) & TEMT_BIT) == 0) ;
         /* wait until previous characters have been transmitted */
outp(LINE_CONTROL_ADDRESS, 0x3B); /* even stick parity */ }
```

4. Wait until bit THRE is set (line status register), then write the lower eight data bits to the transmitter holding register.

```
void SendChar(unsigned char lowerEightBits)
{
       while ((inp(LINE_STATUS_ADDRESS) & THRE_BIT) == 0) ;
         /* wait until the transmitter holding register is empty */
outp(TRANSMIT_BUFFER_ADDRESS, lowerEightBits);
 }
```

5. Wait until TEMT is set, indicating that transmission is complete, then return to receive mode: clear the transmit enable bit (modem control register) and go into even stick parity mode (line control register).

```
 void ReceiveMode(void)
 {
       while ((inp(LINE_STATUS_ADDRESS) & TEMT_BIT) == 0) ;
               /* wait until transmission is complete */
       outp(MODEM_CONTROL_ADDRESS, 0);
               /* clear transmit enable bit, whatever it is */
       outp(LINE_CONTROL_ADDRESS,0x3B);
               /* return to even-stick-parity mode */
 }
```

APPENDIX A

SAMPLE PROGRAM

The following C-language program uses an extremely simple nine-bit protocol to allow two or more NBS-2 equipped PC's to "talk" to each other over a single shielded twisted pair cable. Reception is done by polling the interrupt identification register, then putting characters into a ring buffer. When time permits, received characters are removed from the ring buffer and displayed (putchar() function).

APPENDIX A: SAMPLE PROGRAM

```
/*
 *     NBS-2 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 ON   (OUT1=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 = "*** 03/19/91 ***";

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

#define BASE_ADDRESS 0x2F8  /* COM2 = 2F8 */
#define TX_ENABLE_BIT OUT1_BIT     /* OUT1_BIT, RTS_BIT, or DTR_BIT */

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

/*** 16550 register addresses ***/
#define DIVISOR_LOW_ADDRESS (BASE_ADDRESS+0)
#define DIVISOR_HIGH_ADDRESS (BASE_ADDRESS+1)
#define TRANSMIT_BUFFER_ADDRESS (BASE_ADDRESS+0)
#define RECEIVE_BUFFER_ADDRESS (BASE_ADDRESS+0)
#define INTERRUPT_ENABLE_ADDRESS (BASE_ADDRESS+1)
#define INTERRUPT_ID_ADDRESS (BASE_ADDRESS+2)
#define FIFO_CONTROL_ADDRESS (BASE_ADDRESS+2)
#define LINE_CONTROL_ADDRESS (BASE_ADDRESS+3)
#define MODEM_CONTROL_ADDRESS (BASE_ADDRESS+4)
#define LINE_STATUS_ADDRESS (BASE_ADDRESS+5)
#define MODEM_STATUS_ADDRESS (BASE_ADDRESS+6)
#define SCRATCH_ADDRESS (BASE_ADDRESS+7)

/*** useful UART register bits ***/
#define DR_INTERRUPT 0x01   /* interrupt enable register */
#define RTS_BIT 0x02        /* modem control register */
#define OUT1_BIT 0x04
#define DTR_BIT 0x01
#define TEMT_BIT 0x40              /* line status register */
#define THRE_BIT 0x20
#define DR_BIT 0x01
#define PE_BIT 0x04

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



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

```
/*** 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 = 56000.0;  /* default, in bits per second (baud) */
double crystalFrequency = 8.0;     /* default, in MHz */
unsigned int thisNodeAddress;

/*** 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 EnableFifoModeIfAppropriate(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-2 talk program, for 2-wire (half-duplex) RS-485.");
       puts(copyright);
       puts(version);

              /*** 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 */
       outp(LINE_CONTROL_ADDRESS,0x3B);
              /* set even stick parity mode, appropriate for reception */

       EnableFifoModeIfAppropriate();
       outp(INTERRUPT_ENABLE_ADDRESS, DR_INTERRUPT);
         /* enable received data available & character timeout 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 ***/
       outp(FIFO_CONTROL_ADDRESS,0);      /* disable FIFO mode */
       outp(INTERRUPT_ENABLE_ADDRESS,0);  /* disable all interrupts */
       outp(MODEM_CONTROL_ADDRESS,0);     /* clear transmit enable bit */
}

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

       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(INTERRUPT_ID_ADDRESS) & 4)) {
                     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) 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 ***/
{
       *ninthBit = ((inp(LINE_STATUS_ADDRESS) & PE_BIT) ? 1 : 0);
       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 ***/
{
       outp(MODEM_CONTROL_ADDRESS, TX_ENABLE_BIT);
              /* set transmit enable bit */
}

void ReceiveMode(void)
/*** AFTER TRANSMISSION IS COMPLETE, DISABLE RS-485 DRIVERS ***/
{
       while ((inp(LINE_STATUS_ADDRESS) & TEMT_BIT) == 0) ;
              /* wait until transmission is complete */
       outp(MODEM_CONTROL_ADDRESS, 0);
              /* clear transmit enable bit, whatever it is */
       outp(LINE_CONTROL_ADDRESS,0x3B);
              /* return to even-stick-parity mode */
}

void SetNinthBit(void)      /* for transmission */
{
       while ((inp(LINE_STATUS_ADDRESS) & TEMT_BIT) == 0) ;
              /* wait until previous characters have been transmitted */
       outp(LINE_CONTROL_ADDRESS, 0x2B); /* odd stick parity */
}

void ClearNinthBit(void) /* for transmission */
{
       while ((inp(LINE_STATUS_ADDRESS) & TEMT_BIT) == 0) ;
              /* wait until previous characters have been transmitted */
       outp(LINE_CONTROL_ADDRESS, 0x3B); /* even stick parity */
}

void SendChar(uchar lowerEightBits)
/* SET OR CLEAR NINTH BIT BEFORE CALLING THIS PROCEDURE */
{
       while ((inp(LINE_STATUS_ADDRESS) & THRE_BIT) == 0) ;
              /* wait until the transmitter holding register is empty */
       outp(TRANSMIT_BUFFER_ADDRESS, lowerEightBits);
}

void EnableFifoModeIfAppropriate(void)
/* FIFO MODE WILL ONLY BE USED IF THE UART IS A 16550AF */
{
       outp(FIFO_CONTROL_ADDRESS,7);
              /* enable and clear transmitter and receiver FIFOs &
                 set receiver FIFO trigger level to 1 byte */
       if ((inp(INTERRUPT_ID_ADDRESS) & 0xF0) == 0xC0)  /* check UART */
              printf("16550AF UART detected: FIFO mode will be used.\n");
       else
              outp(FIFO_CONTROL_ADDRESS,0);      /* disable FIFO mode */
}



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) {
              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 / (16e-6 * baudRate);
       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;
}
```