

Buffered Asynchronous Communications Controller User Manual

(Model 2227B or Option 62)

© Wang Laboratories, Inc., 1978



LABORATORIES, INC.

ONE INDUSTRIAL AVENUE, LOWELL, MASSACHUSETTS 01851, TEL. (617) 459-5000, TWX 710 343-6769, TELEX 94-7421

Disclaimer of Warranties and Limitation of Liabilities

The staff of Wang Laboratories, Inc., has taken due care in preparing this manual; however, nothing contained herein modifies or alters in any way the standard terms and conditions of the Wang purchase agreement, lease agreement, or rental agreement by which this equipment was acquired, nor increases in any way Wang's liability to the customer. In no event shall Wang Laboratories, Inc., or its subsidiaries be liable for incidental or consequential damages in connection with or arising from the use of this manual or any programs contained herein.

WANG

LABORATORIES, INC.

ONE INDUSTRIAL AVENUE, LOWELL, MASSACHUSETTS 01851, TEL. (817) 459-5000, TWX 710 343-8769, TELEX 94-7421

PREFACE

Information regarding the installation and operation of the Buffered Asynchronous Communications Controller is provided in this manual.

Chapter 1 describes the features of the controller and some important modem considerations for communications applications. Chapter 2 describes programming techniques related to the controller.

Readers of this manual should be familiar with the BASIC language capabilities and general programming techniques of the Wang system being used in conjunction with the Buffered Asynchronous Communications Controller.

,

CONTENTS

	Page
CHAPTER 1	CONTROLLER FEATURES
1.1	General Information 1
1.2	Installation 3
	Modem Considerations 3
1.3	Connector Pin Assignments 4
1.4	Controller and Modem Interaction 5
1.5	Asynchronous Transmission/Reception 6
	Character Transmission 7
	Character Reception 7
1.6	Data Buffering 8
1.7	Substitution for Characters Received in Error 9
1.8	Transmission Delays Following Specified Characters 9
1.9	Code Translation 9
1.10	Insertion and Removal of Shift Characters 10
1.11	Detecting End-of-Record Characters 11
1.12	Monitoring Received Timeouts 12
1.13	Sending and Detecting Break Signals 12
CHAPTER 2	PROGRAMMING TECHNIQUES
2.1	General Considerations 14
2.2	Specifying the Communications Control Vector 14
2.3	The Communications Status Vector 19
2.4	CPU and Controller Interaction via \$GIO Statements 20
2.5	An Example 27
APPENDIX A	ASCII CODE SET 32
APPENDIX B	SPECIFICATIONS 33
INDEX 34
EQUIPMENT MAINTENANCE 37
CUSTOMER COMMENT FORM Last Page

TABLES

	Page
Table 1-1. Connector Pin Assignments	4
Table 2-1. Valid Communications Control Vector Specifications . . .	17
Table 2-2. Communications Status Vector Information	19
Table 2-3. Microcommand Sequences for Controller and CPU Interaction	20
Table A-1. ASCII Code	32

FIGURES

	Page
Figure 1-1. Asynchronous Data Transmission	6
Figure 2-1. Communications Control Vector Format	16



CHAPTER 1 CONTROLLER FEATURES

1.1 GENERAL INFORMATION

Wang's Buffered Asynchronous Communications Controller is available in physically different but operationally equivalent versions. One version, called the Model 2227B, is a double-card controller attached to a mounting bar for plug-in compatibility with the I/O slots in a 2200 Series CPU (Central Processing Unit). Another version of the controller, called Option 62, has no mounting bar and is configured to fit within the housing of compact units such as the PCS and PCS-II. A copy of this manual is provided with each Buffered Asynchronous Communications Controller installed in a Wang system.

For program control of data transmission and reception via the communications controller, the \$GIO statement is required. The \$GIO statement is standard or available as an option in most 2200 Series central processors; also, the statement is standard in PCS and PCS-II units.

A 3.6-m (12-ft) Wang-supplied cable and a 25-pin EIA (Electronic Industries Association) RS-232-C, CCITT V.24 compatible connector (on the Model 2227B mounting bar or on the PCS back cover) facilitate hookup of a modem, i.e., a modulator/demodulator. A modem is needed for communications applications since data signals from a computer must be converted (modulated) into a range of frequencies suitable for transmission over telephone lines; similarly, data signals received via telephone lines must be demodulated before transfer to a computer. Modem considerations related to the Buffered Asynchronous Communications Controller are presented in Sections 1.2 and 1.4.

Though primarily designed for communications applications, the controller is well-suited for direct connection of RS-232-C compatible asynchronous transmission equipment such as the following: local CRT terminals, graphic display terminals, analytical equipment and laboratory instrumentation in general. For hookup of such equipment, a null modem (available from Wang Laboratories) may be required.

The controller has an integrated microprocessor and multicharacter input/output buffers to simplify telecommunications control procedures and reduce CPU processing requirements. Fixed microcode, residing in the controller's read-only-memory, implements standard and optional operations such as the following:

- . data buffering

- . code translation
- . substitution for characters received in error
- . communications control, e.g., monitoring CPU ready/busy conditions, monitoring modem signals, implementing line turnaround procedures
- . break signal detection and transmission
- . detection of received timeouts
- . detection of end-of-record characters
- . automatic insertion and removal of shift characters
- . automatic transmission delays following several specified characters
- . sensing the Secondary Received Line Signal Detector and setting the Secondary Request to Send signal (also called reverse or supervisory channel data signals).

The standard and special features of the controller enable a Wang system to be programmed to transmit and receive data using the line discipline of a variety of asynchronous CRT and mechanical printer terminals.

The controller's 1K-byte random-access-memory is used for storage of initialization information (including code translation tables and a communications control vector), storage of current status information, and input/output data buffering. The desired transmission rate, communication mode, character format options, and special operations for a particular application are selected under program control by specified values in a communications control vector. The vector is loaded into the controller by a \$GIO statement in the user's application program operating in the CPU.

A selectable-speed clock on the controller supports serial asynchronous transmission and reception at line speeds from 50 to 9600 bits per second (bps). Any one of the following rates can be set via the initializing control vector: 50, 75, 100, 110, 134.5, 150, 200, 300, 600, 1200, 1800, 2400, 3600, 4800, 7200 or 9600 bps. Usually, rates from 50 to 1800 bps are used for point-to-point, dial-up telecommunications applications while rates above 1800 bps are used for directly connected RS-232-C compatible equipment.

The communications controller supports any one of the following transmission modes:

- . half-duplex mode (independent transmission one-way-at-a-time alternately)
- . half-duplex mode with automatic deletion of received null characters
- . full-duplex mode (independent transmission two-ways simultaneously)
- . full-duplex mode with automatic deletion of received null characters.

The desired mode is set via a particular byte-position in the communications control vector.

Via other byte-positions in the communications control vector, the character format can be selected from the following options:

- . parity -- odd, even, or no parity
- . number of data bits per character -- 5, 6, 7 or 8
- . number of stop bits per character -- 1, 1.5 or 2.

The features of the Buffered Asynchronous Communications Controller are numerous; descriptions of particular features are presented in this chapter. Programming techniques are presented in Chapter 2, where the format of the communications control vector is shown byte-by-byte in Figure 2-1 and valid values representing the asynchronous transmission options are presented in Table 2-1; also, the format of the communications status vector is given, and \$GIO microcommand sequences needed to operate the controller are supplied. An example is included to illustrate some of the programming techniques.

1.2 INSTALLATION

Installation of a communications controller is the responsibility of a Wang Service Representative. If a controller arrives as an addition to existing equipment, call the Wang Service Representative; do not attempt to install the controller--any such attempt may void the warranty.

After the controller is inspected, diagnostically checked, and installed, one end of the cable supplied with the controller is plugged into the connector attached to the controller. The other end of the cable has an RS-232-C compatible male plug.

If the controller is to be used for point-to-point, dial-up asynchronous telecommunications applications, the other end of the cable should be plugged into a suitable modem. Installation of a modem is not the responsibility of a Wang Service Representative.

Alternatively, if the controller is to be used to interface RS-232-C compatible asynchronous transmission equipment such as a non-Wang CRT terminal or a laboratory instrument, the other end of the cable may need to be plugged into a null modem (available from Wang Laboratories) or may be suitable for direct connection to the non-Wang equipment. Installation or interfacing of non-Wang equipment is not the responsibility of a Wang Service Representative; therefore, information regarding the controller's connector pin assignments and voltage levels is given in Section 1.3.

Modem Considerations

The modem used with the communications controller may be rented from the telephone company serving the locality where a Wang system is installed or may be purchased from any one of several modem vendors. In either case, arrangements should be made with the telephone company for installation of a modem since modems purchased from a vendor must be connected to the telephone network via telephone company installed data access arrangements (DAA) consisting of a telephone handset and modem interface rented from the telephone company.

Before a telephone company representative arrives to install a modem, the location of the Wang system must be planned to ensure its proximity to the telephone equipment. Normally modems or DAA's are wired permanently to a wall; in such cases, subsequent relocation of the Wang system any great distance would necessitate having the telephone company relocate the modem or DAA. The RS-232-C standard recommends use of short cables (less than 50 feet or 15 meters) between data terminal equipment and communications equipment. Longer cable distances are possible for operations at a lower range of transmission rates in an environment relatively free of electromagnetic interference.

Other modem considerations are discussed in Section 1.4.

1.3 CONNECTOR PIN ASSIGNMENTS

Information in this section is provided for readers responsible for interfacing non-Wang equipment to a Wang system via the Buffered Asynchronous Communications Controller. Other readers may wish to ignore the section.

The controller conforms to the nationally recognized EIA RS-232-C and the internationally recognized CCITT V.24 standards for voltage levels and pin connections. The signal polarity and the voltage of driven and detected signals are as follows:

<u>Logic Level</u>	<u>Applied Voltage</u>	<u>Detected Voltage</u>
0 or ON (Spacing)	+8 vdc	+5 to +15 vdc
1 or OFF (Marking)	-8 vdc	-5 to -15 vdc

The pin assignments are listed in Table 1-1 with both the EIA and the CCITT designations given for the circuit associated with each pin. Also, the signal descriptions and sources are included in the table.

Table 1-1. Connector Pin Assignments

Pin	EIA	CCITT	Signal Description	Source
1	AA	101	Protective Ground	
2	BA	103	Transmitted Data	Controller
3	BB	104	Received Data	Modem
4	CA	105	Request to Send	Controller
5	CB	106	Clear to Send	Modem
6	CC	107	Data Set Ready	Modem
7	AB	102	Signal Ground	
8	CF	109	Received Line Signal Detector	Modem
9				
10				
11	SCA	120	Secondary Request to Send	Controller
12	SCF	122	Secondary Rec'd Line Sig. Det.	Modem
13*	SCB	121	Secondary Clear to Send	Modem
14*	SBA	118	Secondary Transmitted Data	Controller
15*	DB	114	Trans. Signal Element Timing	Modem
16*	SBB	119	Secondary Received Data	Modem
17*	DD	115	Receiver Signal Element Timing	Modem
18*		124	Select Frequency Groups	Controller
19*	SCA	120	Secondary Request to Send	Controller
20	CD	108.2	Data Terminal Ready	Controller
21*	CG	110	Signal Quality Detector	Modem
22*	CE	125	Ring Indicator	Modem
23*	CH/CI	111/112	Data Signalling Rate Selector	Controller/Modem
24*	DA	113	Trans. Signal Element Timing	Controller
25			Unassigned	

* Signals not utilized by controller.

1.4 CONTROLLER AND MODEM INTERACTION

The microprocessor on the communications controller can sense the value of the following modem signals:

- . Received Data on Pin 3
- . Clear to Send on Pin 5
- . Data Set Ready on Pin 6
- . Received Line Signal Detector on Pin 8
- . Secondary Received Line Signal Detector on Pin 12.

The microprocessor can set the level of the following modem signals:

- . Data Terminal Ready on Pin 20
- . Request to Send on Pin 4
- . Transmitted Data on Pin 2
- . Secondary Request to Send on Pins 11 and 19.

Any Bell 103 or 202 type modem, or equivalent, can be used with the controller. Very likely, other RS-232-C compatible modems commonly used with asynchronous terminals having transmission rates in the range covered by the controller may prove suitable.

Normally, a 103A type modem is used for operations requiring line transmission speeds up to 300 bits per second. The modem should be ordered with optional features which provide an originate and an answer capability. Also, a receive long space disconnect feature is desirable for use with the controller, but no special features are necessary for a break signal capability.

Normally, a 202C or 202S type modem is used for operations requiring line transmission speeds at 1200 bits per second (and occasionally at lower speeds). Options should include originate, answer, and receive long space disconnect features. Furthermore, if use of a break signal is desired (as is the case more often than not), the reverse channel option should be included.

NOTES:

1. Modems used at both ends of a point-to-point, dial-up communications link must be of similar type. For example, if a Bell 103 type modem is used at one end, another Bell 103 type modem or an equivalent modem must be used at the other end (not a Bell 202 type modem).
2. If acoustic couplers are used at both ends of a communications link, one must have the "originate" feature and the other must have the "answer" feature -- ideally each should have both features.

1.5 ASYNCHRONOUS TRANSMISSION AND RECEPTION

Generally speaking, in asynchronous transmission (often called start-stop transmission) each character is framed by start and stop elements as shown in Figure 1-1. The start element is represented by a transition from a logic "1" voltage level to a logic "0" voltage level. The nominal interval during which the logic "0" level is maintained for a start element is the same length as the interval used for each data bit.

Depending upon the design of the transmitting equipment, the data-bit interval is a fixed value or one of several possible values if the transmission rate for the equipment is selectable. Immediately following transmission of a start element, the voltage level is changed or not changed depending upon whether the first data bit is 1 or 0. (See Figure 1-1.) Similarly, after the first data-bit interval, the voltage level is changed or not, as required, to represent the second data-bit -- and so on successively for each data bit. The number of data bits transmitted is a fixed value or one of several possible values, depending upon the equipment being used.

After the last data bit is transmitted, a parity bit may be transmitted if provisions for parity information are included in the equipment design. The parity bit interval is the same length as the data bit and start bit intervals. The voltage level may be a logic "0" or "1" depending upon the type of parity (odd or even) and also upon the number of 1's occurring in the preceding data bits.

Finally, the stop element is transmitted using a logic "1" voltage level which is maintained until the next character is transmitted. Usually, there is no upper limit to the length of a stop element; however, there is a lower limit, a fixed value or one of several possible values, depending upon the design of the transmitting equipment.

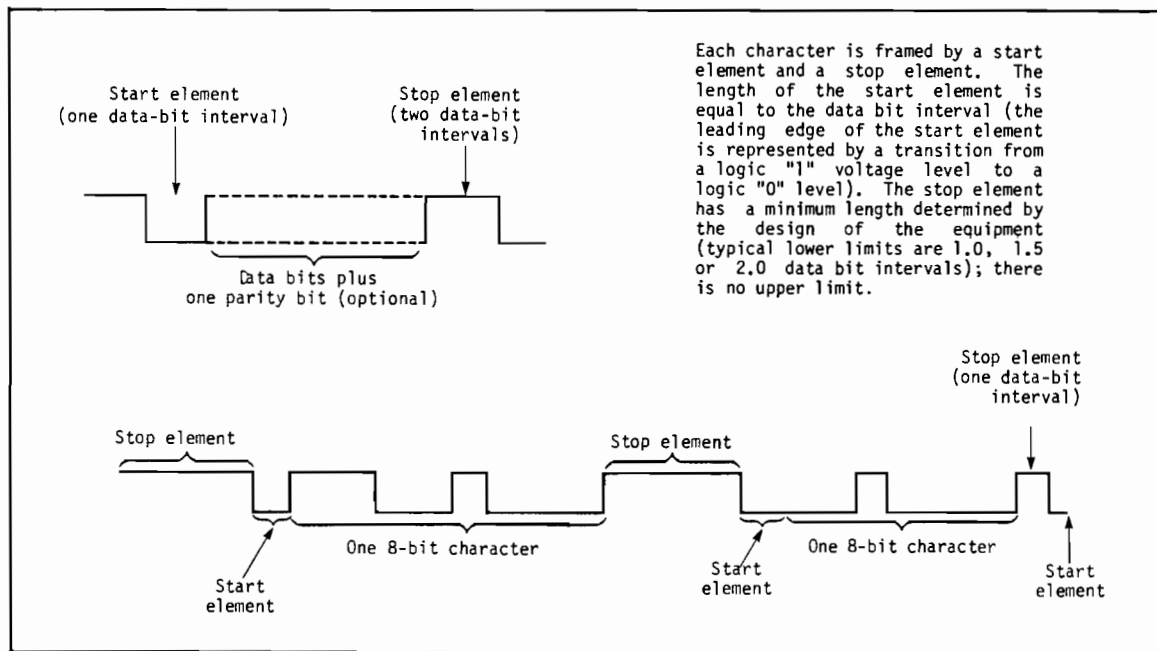


Figure 1-1. Asynchronous Data Transmission

Character Transmission

Wang's Buffered Asynchronous Communications Controller transmits each character by modulating the Transmitted Data signal on Pin 2 in the connector as follows:

1. The Transmitted Data signal is set to "0" for one bit-time, representing the start bit.
2. Successively, low-order bit first, the signal is set for one bit-time to the value of each data bit until the number of transmitted data bits equals the number specified in the communications control vector; therefore, only the low-order 5, 6, 7 or 8 bits of a character are transmitted.
3. If odd or even parity is specified in the communications control vector, the signal is set for one bit-time to the appropriate value for the type of parity specified. In particular, if odd parity is specified, the parity bit is equal to 1 when the preceding data bits contain an even number of 1-bits; thus, for odd parity, the total number of 1's in the data bits plus the parity bit is an odd number. If even parity is specified, the parity bit is equal to 1 when the preceding data bits contain an odd number of 1-bits; thus, for even parity, the total number of 1's in the data bits plus the parity bit is an even number. If no parity is specified, Step 3 is omitted.
4. The Transmitted Data signal is set to "1" for a minimum interval equal to 1, 1.5 or 2 bit-times depending upon the number of stop bits specified in the communications control vector.

When no character is being transmitted, the Transmitted Data signal on Pin 2 is held at the value "1".

Character Reception

The communications controller receives a character by detecting changes in the Received Data signal on Pin 3 in the connector as follows:

1. A transition from the voltage level representing logic "1" to the level representing logic "0" for at least one-half a bit-time is interpreted as the leading edge of the start bit for an incoming character.
2. The Received Data signal is sampled successively at times corresponding to the nominal center of each data bit. In particular, the nominal center of the first data bit is 1.5 bit-times after the leading edge of the start bit. The center of each subsequent bit occurs one bit-time after the center of its predecessor. Successively, low-order bit first, the bits in the character being received are set to correspond to the sampled values. The number of data bit samples taken by the controller equals the number of data bits specified in the communications control vector. If the number of samples is less than 8, the remaining high-order bits in the received character are automatically set to 0 (unless the shift character option is in effect).

3. A parity bit, if specified, is read by sampling the Received Data signal again -- one bit-time after the last data-bit is sampled. The sampled parity value is compared with a calculated value based on the received data bits and the type of parity specified. If the received and calculated parity values are unequal, a designated bit in the communications status vector is set to 1 to indicate a parity error has occurred.
4. One bit-time after the Received Data signal is sampled for a parity value (or for the last data bit if a no-parity option is in effect), the signal is sampled again. Now, if the signal is "1", a valid stop bit is recognized. On the other hand, if the signal is "0", a framing error has occurred and a designated bit in the status vector is set to 1.

NOTE:

For each application, the transmission rate, number of data bits, type of parity, and number of stop bits specified for the communications controller must match the specifications for equipment in use at the other end of a communications link.

1.6 DATA BUFFERING

The controller has two multicharacter data buffers, a 175-byte transmit buffer and a 255-byte receive buffer. With these buffers, data transmission/reception operations performed by the controller with respect to the modem can overlap data input/output operations performed by the CPU with respect to the I/O peripherals designated for a communications application.

For example, after the CPU sends a data string to the controller, the CPU is free to perform an independent task such as fetching the next string of data to be transmitted from the input device -- while the controller is performing such tasks as code translation, character formatting, and transmission to the modem.

NOTE:

If the transmit buffer becomes full while the CPU is sending data to the buffer, the data transfer rate from the CPU to the controller automatically slows to the rate at which characters are being transmitted from the buffer. No characters are lost.

On the other hand, the controller is free to receive a data string, perform such operations as code translation, and store the data in the receive buffer -- while the CPU is performing an independent task such as outputting data to a designated peripheral.

NOTE:

If characters are received when the receive buffer is full, a buffer overrun condition occurs and the appropriate error bit is set in the communications status vector. No other action is taken by the controller.

1.7 SUBSTITUTION FOR CHARACTERS RECEIVED IN ERROR

When a character is received with either a parity or a framing error, a substitute character (defined by byte 4 in the communications control vector) is automatically supplied by the controller, and the appropriate error bit is set in the communications status vector. For example, a special character such as @, or any other character not likely to occur in the incoming data, can be specified as the character to automatically replace any characters received in error. Replacement occurs before code translation, if any, is performed.

1.8 TRANSMISSION DELAYS FOLLOWING SPECIFIED CHARACTERS

When sending data to a mechanical printer terminal such as an IBM 2741 or a Wang 1200, time delays are needed after transmission of TAB and CR (carriage return) characters to allow the printing element sufficient time to reach the proper position without loss of subsequent characters.

A special feature of the controller removes the necessity for the user's application program operating in the CPU to introduce time delays following transmission of special characters. Bytes 11 through 18 of the communications control vector can be used, in four two-byte groups, to define up to four special characters and the transmission delay associated with each special character. During data transmission, the controller automatically delays for the specified time following transmission of any character matching one of the specified characters.

1.9 CODE TRANSLATION

The code translation feature of the controller allows data interchange between the CPU and the controller in the ASCII code (American Standard Code for Information Interchange) native to Wang's System 2200 and PCS -- regardless of the transmission/reception code for a particular application.

Space is reserved in the controller for two 256-byte code translation tables, a transmit-code translation table and a receive-code translation table. Specification of such tables is optional. Translation tables, supplied in the user's application program operating in the CPU, must be loaded into the controller by appropriate \$GIO statements in the program. (See Section 2.4.)

The automatic code translation operation is enabled by loading a transmit or a receive code translation table (or both) after loading the communications control vector. If no tables are loaded, the code translation feature is effectively disabled.

During transmission, a character sent from the CPU to the controller becomes an 8-bit index for a table lookup in the transmit-code translation table. An 8-bit character obtained from the table is placed in the transmit buffer; however, if byte 3 of the communications control vector specifies less than 8 data bits per character, only the relevant low-order bits of each character are actually transmitted.

During reception, a character received by the controller is used as an 8-bit index for a table lookup in the receive-code translation table, and an 8-bit character is obtained from the table. If the translated character is a null character, i.e., $(00)_{16}$, and the high-order hexdigit in byte-position-2 in the communications control vector has the value 1 or 4, the character is discarded; otherwise, the translated character is placed in the receive buffer.

NOTE:

Superfluous characters used for timing or fill can be removed automatically by translating them to null characters. This feature is applicable to half-duplex and full-duplex operations. See Table 2-1.

1.10 INSERTION AND REMOVAL OF SHIFT CHARACTERS

For applications involving data transmission and reception in a code set which utilizes shift characters, e.g., a Baudot code set, an IBM 2741 code set, or a Wang 1200 code set, a special feature of the communications controller removes the necessity for the user's program operating in the CPU to handle insertion and removal of shift characters. Instead, the upshift and the downshift characters are defined in bytes 7 and 8 of the communications control vector. The number of data bits per character is set to 5 or 6 (depending upon the application) by choosing an appropriate value for the high-order hexdigit in byte 3 of the communications control vector. Then, to activate automatic insertion and removal of shift characters, code translation tables are suitably defined and loaded into the controller.

During data transmission, the controller examines and interprets the two high-order bits of each translated character in the transmit buffer as "shift status" bits as follows:

<u>Two High-order Bits</u>	<u>Meaning</u>
00	Downshifted character.
01	Upshifted character.
10	Character doesn't care about shift status.
11	Character doesn't care about shift status.

A shift character is automatically transmitted between any two characters having different "shift status" bits. In such cases, if the second character has upshifted status, an upshift character is transmitted prior to transmission of the second character; alternatively, if the second character has downshifted status, a downshift character is transmitted prior to transmission of the second character. The shift-status bits are not transmitted since the number of data bits being transmitted per character is only the low-order 5 or 6 bits if byte 3 in the communications control vector has been appropriately specified.

During reception, the controller sets the value of the high-order eighth bit of each received character (before code translation) according to the most recently received shift character as follows:

<u>High-order Bit</u>	<u>Meaning</u>
1	Upshifted character.
0	Downshifted character.

NOTE:

If a received character is a shift character, the character is discarded; otherwise, the character is code translated and placed in the receive buffer.

1.11 DETECTING END-OF-RECORD CHARACTERS

The end-of-record detection feature is convenient for applications where a received data stream contains meaningful record delimiters, e.g., CR (carriage return) codes. The feature is particularly convenient for applications where there is no necessity to display the data while being received.

Any number of characters can be defined as end-of-record characters, thereby permitting the controller to divide a received data stream into records while eliminating the need for the user's program operating in the CPU to perform the task.

To activate the end-of-record detection feature, byte 6 in the communications control vector must be set to HEX(01); also, a suitably defined receive-code translation table must be loaded into the controller. To be suitably defined, the high-order bit for codes in the receive-code translation table must be set to 1 for each character defined as an end-of-record character (and set to zero for all other characters).

During reception, if end-of-record detection is enabled, the controller maintains a count of the number of end-of-record characters currently stored in the receive buffer. The count is maintained in byte 5 of the communications status vector (see Table 2-2).

With an appropriate \$GIO statement (see Section 2.4), the user's application program can read the status vector into the CPU and subsequently test the status information to ensure the availability of a complete record before requesting transfer of buffered data via a \$GIO statement which performs data transfer from the controller to the CPU.

When data is actually transferred from the controller's receive buffer to the CPU, only those characters up to (and including) the first end-of-record character are transferred. Furthermore, the high-order bit in the end-of-record character is changed from 1 to 0 when the character is transferred.

NOTE:

If the end-of-record detection feature is not needed for an application (or cannot be utilized because the high-order bit for codes in the receive-code translation table is set to 1 for a purpose other than defining an end-of-record character), byte 6 in the communications control vector should be set to HEX(00) to disable end-of-record detection.

1.12 MONITORING RECEIVED TIMEOUTS

The communications controller has the capability to monitor received timeouts. Byte 5 in the communications control vector is used to set the binary value of the timeout in units of 0.1 seconds. For example, the minimum timeout condition, 0.1 seconds, is specified by storing HEX(01) in byte-position-5. The maximum timeout condition, 25.5 seconds, is specified by storing HEX(FF) in byte-position-5. On the other hand, storing HEX(00) in byte-position-5 disables the monitoring feature for received timeouts.

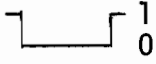


NOTE:

If a timeout interval is specified, the controller maintains a received data timeout countdown in byte 6 of the communications status vector (see Section 2.3).

1.13 SENDING AND DETECTING BREAK SIGNALS

The communications controller has the capability to send and detect break signals under control via the user's program operating in the CPU. Bytes 9 and 10 in the communications control vector are used to define the break signal transmission and detection intervals, respectively, in units of 10 milliseconds. For example, HEX(14) stored in byte-position-9 defines an interval equal to 200 milliseconds for transmitted break signals. Similarly, HEX(11) stored in byte-position-10 defines an interval equal to 170 milliseconds for detection of break signals.

In addition to specifying the break signal intervals in bytes 9 and 10 of the communications control vector, it is necessary to use the low-order hexdigit position in byte 2 to specify the participating modem signals and the polarity of the break signals as follows:

<u>Byte 2 (Low-order Hexdigit)</u>	<u>Break Signal Polarity</u>	<u>Modem Signals</u>
0	none	none
1		Transmitted/Received Data
2		Secondary Request to Send, or Secondary Received Line Signal Detector
3		Secondary Request to Send, or Secondary Received Line Signal Detector

NOTES:

1. The Transmitted Data and Received Data modem signals are used with Bell 103 type modems.
2. Normally the Secondary Request to Send and Secondary Received Line Signal Detector modem signals are used with Bell 202 type modems which must be ordered with the reverse channel option in order to support break signal operation.

Transmission of a break signal by the controller involves inverting the level of the specified modem signal (i.e., Transmitted Data or Secondary Request to Send) for an interval defined by byte 9 of the communications control vector.

Detection of a break signal occurs when the controller senses the level of a specified modem signal (Received Data or Secondary Received Line Signal Detector) is being continuously inverted for an interval at least as long as the interval defined by byte 10 of the communications control vector.

NOTE:

Detection of a break signal causes the "break signal received" bit in the status vector to be set. No other action is taken by the controller.

CHAPTER 2 PROGRAMMING TECHNIQUES

2.1 GENERAL CONSIDERATIONS

When writing an application program for the Buffered Asynchronous Communications Controller, a programmer should organize the program in distinct modules designed to achieve initialization and communications functions. The communications module could overlay the initialization module or, if preferred, the two modules could coexist in memory.

Generally speaking, an initialization module defines the 20-byte communications control vector and assigns particular values to byte-positions denoting information such as the desired transmission rate in bits per second, the number of data bits per character, the type of parity (if any), and the number of stop bits per character. Other values in the control vector indicate whether the controller is to activate such features as break detection, monitoring of received timeouts, half or full duplex operation with or without automatic removal of null characters, and substitution of a special character for characters received with parity or framing errors. The module also supplies the \$GIO statement needed to load the control vector into the controller.

An initialization module also defines transmit and receive code translation tables, if required for the application, and supplies the \$GIO statements needed to load such tables into the controller. Additionally, the initialization module might define and initialize variables required by the communications module even though the variables are stored in the CPU rather than the controller memory.

2.2 SPECIFYING THE COMMUNICATIONS CONTROL VECTOR

The format of the communications control vector is shown in Figure 2-1, and valid specifications for the vector are given in Table 2-1. The table is divided into two portions since the first three bytes of the vector are dual purpose while the remaining bytes are single purpose with respect to the available communications options and features.

In the application program residing in the CPU, the control vector should be defined by a one-dimensional array having 18 or 20 elements with one byte per element. For example, to represent the control vector by the array C\$(), use

```
DIM C$(20)1
```

Also, as a general programming practice, all elements should be initialized to binary zero by a statement of the form

```
INIT(00) C$()
```

before assigning values to particular elements in the array.

Then, as illustrated by the following statements, individual byte-positions in the control vector can be assigned values other than binary zero to select the desired combination of options and define any special characters for the application.

<u>Statement</u>	<u>Comment</u>
C\$(1) = HEX(17)	One stop bit; 300 bits per second.
C\$(2) = HEX(11)	Half-duplex with automatic deletion of null characters; break enabled on transmit/receive.
C\$(3) = HEX(23)	Seven data bits; odd parity.
C\$(4) = HEX(5E)	Substitute character for parity or framing error is an up-arrow, ↑.
C\$(5) = HEX(0A)	Timeout interval is 1 second.

Care must be exercised when defining some special characters to choose a compatible value in the first three byte-positions of the communications control vector. For example, if defining upshift and downshift characters in bytes 7 and 8, the high-order hexdigit in byte 3 must have the value 0 or 1 (since the shift feature can be used only with code sets having 5 or 6 data bits per character). See Section 1.10 and Table 2-1.

NOTE:

The communications control vector must be loaded into the controller via a \$GIO statement having the appropriate microcommand sequence from Table 2-3. See Section 2.4.

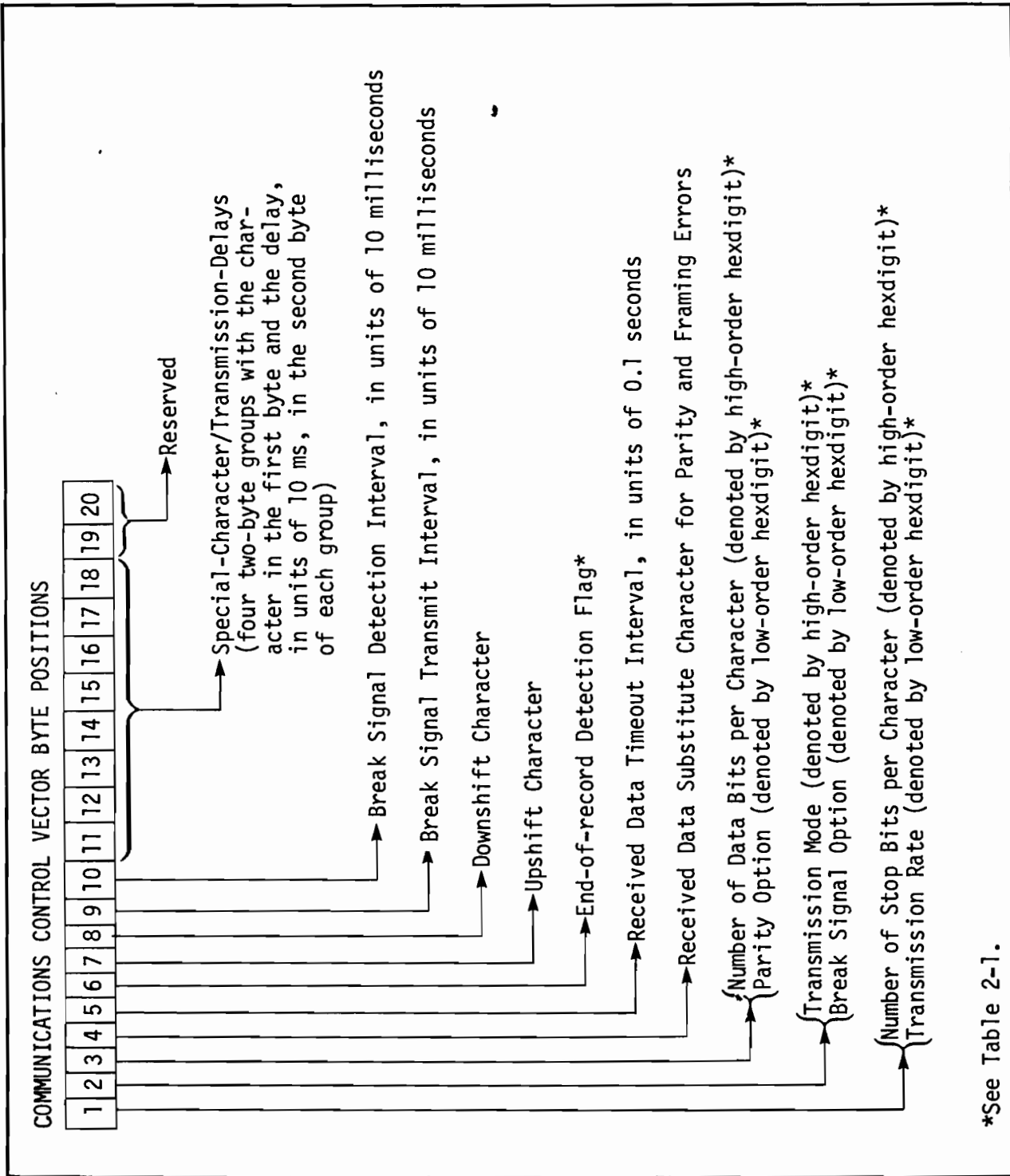


Figure 2-1. Communications Control Vector Format

Table 2-1. Valid Communications Control Vector Specifications

Byte*	High-order Hexdigit	Low-order Hexdigit
1	0 = Illegal value 1 = 1 Stop bit 2 = 1.5 Stop bits 3 = 2 Stop bits	0 = 50 bits per second 1 = 75 bps 2 = 100 bps 3 = 110 bps 4 = 134.5 bps 5 = 150 bps 6 = 200 bps 7 = 300 bps 8 = 600 bps 9 = 1200 bps A = 1800 bps B = 2400 bps C = 3600 bps D = 4800 bps E = 7200 bps F = 9600 bps
2	0 = Half duplex 1 = Half duplex with deletion of received null characters 2 = Full duplex 3 = Full duplex with deletion of received null characters	0 = Break disabled 1 = Break enabled on transmit/receive 2 = Break enabled on Secondary Req. to Send & Sec. Rec. Line Sig. Det. 3 = Same as 2 with inverted polarity
3	0 = 5 Data bits per character 1 = 6 Data bits 2 = 7 Data bits 3 = 8 Data bits	0 = No parity 1 = Even parity 2 = No parity 3 = Odd parity

*See Table 2-1 (Continued) for bytes 4 through 20.

Table 2-1. Valid Communications Control Vector Specifications (Continued)

Byte	High and Low Order Hexdigits*	Remarks
4	xy = Substitute character for parity/framing errors.	Each received character having a parity or framing error is replaced by the designated character (replacement occurs prior to code translation if translation tables are being used). See Section 1.7.
5	xy = Timeout interval in units of 0.1 seconds.	The specification in hexadecimal notation represents the timeout interval in units of 0.1 seconds, e.g., $(24)_{16} = (36)_{10}$ specifies an interval of 3.6 seconds. See Section 1.12.
6	00 = Disable end-of-record detection. 01 = Enable end-of-record detection.	If enabled, the end-of-record characters must be defined via the receive-code translation table by setting the high-order bit to 1 for each code corresponding to an incoming end-of-record character. See Section 1.11.
7	xy = Upshift character.	To enable shift code insertion/deletion, the high-order hexdigit in byte 3 of the control vector must be 0 or 1 (i.e., the number of data bits per character must be 5 or 6). Also, the transmit-code translation table must identify all downshifted, upshifted, and "don't care" characters by setting the two high-order bits to 00, 01, and 10 or 11 as described in Section 1.10. The receive-code translation table must allow for the controller's automatic setting (before translation) of the high-order bit to 1 for all incoming upshifted characters.
8	xy = Downshift character.	
9	xy = Break signal transmit interval in units of 10 ms.	To enable break signal transmission/detection, the low-order hexdigit in byte 2 of the control vector must specify the polarity and the modem signals. If bytes 9 and 10 are both HEX(00), the low-order hexdigit in byte 2 should be 0. The byte 9 and 10 specifications in hexadecimal notation represent break signal transmit and receive intervals in units of 10 milliseconds, e.g., $(12)_{16} = (18)_{10}$ specifies a 180 ms interval. See Section 1.13.
10	xy = Break signal detection interval in units of 10 ms.	
11	xy = Special character.	Up to four special characters can be defined in bytes 11, 13, 15 and 17. The delay associated with a particular character is specified in the following byte, in hexadecimal notation representing the interval in units of 10 milliseconds. The maximum delay, HEX(FF), is 2.5 seconds. During transmission, the controller automatically delays for the specified time following the transmission of one of the specified characters. If a transmit-code translation table is used, each special character must be defined with respect to its code after translation. If the automatic transmission delay capability is not desired, bytes 11 through 18 should each be set to HEX(00).
12	00 = No transmission delay. xy = Transmission delay in units of 10 ms.	
13	Same as byte 11.	
14	Same as byte 12.	
15	Same as byte 11.	
16	Same as byte 11.	
17	Same as byte 11.	
18	Same as byte 12.	

*x and y each denote any hexdigit (0 through 9, A through F). If a feature is not desired, the byte positions associated with the feature can be ignored if the communications control vector has been initiated to binary zero.

2.3 THE COMMUNICATIONS STATUS VECTOR

Space is reserved in the random access memory of the controller for a communications status vector whose byte and bit positions are used automatically as shown in Table 2-2. The first three bytes of the status vector are cleared automatically whenever it is read and when the communications control vector is loaded into the controller from the CPU. See Section 2.4.

Flags are set in particular bit positions in the first three bytes of the status vector during controller operation. In bytes 4 and 5, the current number of characters in the receive buffer and the number of end-of-record characters are maintained as binary counts. Similarly, in byte 7, the current number of characters in the transmit buffer is maintained. Byte 6, on the other hand, is similar to a real time clock whose value is initiated to the timeout interval specified in byte 5 of the communications control vector each time one of the following events occurs:

- a) a \$GIO "start receiving data" operation begins,
- b) a line turnaround occurs during a \$GIO "send, then receive data" operation, or
- c) a character is received during either operation.

However, if the value in byte 6 of the status vector is not reset by one of these operations, the countdown proceeds to zero.

Whenever desired, the information in the status vector can be read (transferred to the CPU) by a \$GIO statement having the appropriate microcommand sequence from Table 2-3. See Section 2.4. After transfer to the CPU, status vector information can be tested, as required, by the application program.

Table 2-2. Communications Status Vector Information

Byte	Bit*	Meaning
1	1	1 = Break signal received
2	1 2 3	1 = Received Line Signal Detector On 1 = Sec. Rec'd Line Sig. Det. On 1 = Data Set Ready modem signal On
3	1 2 3	1 = Receive parity error detected 1 = Receive buffer overrun error detected 1 = Receive framing error detected
4	all	Binary count of the number of characters in the receive buffer.
5	all	Binary count of the number of end-of-record characters in the receive buffer.
6	all	Received data timeout countdown.
7	all	Binary count of the number of characters in the transmit buffer.

*Bit positions in each byte are numbered from 1 (low-order) to 8 (high-order).

2.4 CPU AND CONTROLLER INTERACTION VIA \$GIO STATEMENTS

To operate the Buffered Asynchronous Communications Controller, the user's application program residing in the CPU should include \$GIO statements with suitable microcommand sequences. A list of valid microcommand sequences for controller operations is presented in Table 2-3.

Table 2-3. Microcommand Sequences for Controller and CPU Interaction

Controller and CPU Interaction	Microcommand Sequence*	Remarks
Set communications control vector	4402 A000 440C	
Read communications status vector	(See ** below)	
Load transmit code translation table	4404 A000 440C	
Load receive code translation table	4405 A000 440C	
Disconnect	4406	
Send break signal	4407	
Start receiving data	4408	For half or full duplex.
Transfer received data to CPU	(See ** below)	For half or full duplex.
Send data	440A A000 440C	For half or full duplex.
Send, then receive data	440B A000 440C	For half duplex only.
Stop transmitting	440C	For full duplex protocols.
Continue transmitting	440D	For full duplex protocols.

*A microcommand sequence can be specified directly or indirectly in a \$GIO statement. If specified directly as the arg-1 component, each four-hexdigit-code can be separated from the previous one by a space for readability as shown in this table. If specified indirectly by assigning the sequence to a variable and including the variable in a statement, spaces cannot be used between the four-hexdigit-codes, e.g., A\$ = HEX(4402A000440C); furthermore, the dimension of the variable must be large enough to ensure the presence of two trailing space characters which serve as the pseudo-microcommand 2020 denoting the end of the sequence. Unpredictable results may occur if at least one trailing blank does not follow an indirectly specified microcommand sequence. (See the General I/O Instruction Set Reference Manual which accompanies each Wang system having the \$GIO statement in its BASIC language instruction set, or see the BASIC-2 Language Reference Manual if using a 2200VP or 2200MVP system.)

**The valid microcommand sequence is dependent upon whether the controller is installed in either a 2200MVP central processor or some other 2200 central processor. See the following table.

Controller and CPU Interaction	Non-2200MVP	Sequence for 2200MVP
Read communications status vector	4403 C620	4403 1020 02FF 03FF 1223 C620
Transfer received data to CPU	4409 C620	4409 1020 02FF 03FF 1223 C620

Brief descriptions of each of the operations in Table 2-3 follow. A sample \$GIO statement is shown for each operation; however, the comments and variables used in the statement may be given different names in a user's program, and the address 01C may not be appropriate for the communications controller in the system being used.

Set Communications Control Vector

```
$GIO SET CCV /01C (4402 A000 440C, G$) C$()
```

The communications control vector (CCV) defined by the array C\$() is set (loaded) into the controller when the statement is executed. Here, 01C is the address of the controller, and G\$ represents the error/status/general-purpose registers.

NOTE:

The controller's transmit and receive buffers, as well as the communications status vector and code translation tables, are cleared automatically when the communications control vector is loaded.

Read Communications Status Vector

```
$GIO READ CSV /01C (4403 C620, G$) A$
```

or for a 2200MVP central processor

```
$GIO READ CSV /01C (4403 1020 02FF 03FF 1223 C620, G$) A$
```

The information currently in the communications status vector is read into the CPU and stored in the character string A\$ (which must be at least 7 bytes long).

NOTE:

The error and received break indicators (i.e., bytes 1 and 3) in the communications status vector are cleared automatically after the status vector information is read into the CPU. In a program for a 2200MVP central processor, insert a line containing a \$BREAK statement before each line containing a "read communications status vector" statement.

Load Transmit Code Translation Table

\$GIO LOAD TTBL /01C (4404 A000 440C, G\$) C1\$()

The transmit code translation table is loaded into the controller from the array C1\$() if such an array is previously defined in the application program. The optional transmit code translation feature is enabled only if a transmit code translation table is loaded after the communications control vector is loaded into the controller.

NOTES:

1. The transmit code translation table should be exactly 256 bytes in length and represent the codes to which System 2200 ASCII characters are to be converted prior to storage in the transmit buffer. The byte positions in the table should contain the "after translation characters" arranged in a sequence corresponding to the "before translation characters", i.e., the System 2200 ASCII characters.
2. If shift character automatic insertion/removal is in effect (i.e., the specified number of data bits per character is 5 or 6), the two high-order bits of each code in the transmit code translation table must conform to the appropriate values given in Section 1.10.

Load Receive Code Translation Table

```
$GIO LOAD RTBL /O1C (4405 A000 440C, G$) C2$()
```

The receive code translation table is loaded into the controller from the array C2\$() if such an array is previously defined in the application program. The optional receive code translation feature is enabled only if a receive code translation table is loaded after the communication control vector is loaded into the controller.

NOTES:

1. The receive code translation table should be exactly 256 bytes long. The byte positions in the table should contain ASCII characters (the "after translation characters" in this case) arranged in a sequence corresponding to the "before translation characters". The translation procedure is equivalent to using the binary equivalent of an incoming character's hexadecimal code as an index for a table look-up operation by which the appropriate translation character is found. For example, if the incoming non-ASCII character is a HEX(18), the binary value is 24; therefore, the corresponding ASCII character should be located in the 25th position of the receive translation table. (Keep in mind that the first position in the table corresponds to the binary value zero.)
2. If shift character automatic insertion/removal is in effect, the 256-byte receive code translation table represents two 128-byte tables. The first 128 byte positions in the table should represent the conversions for incoming downshifted characters corresponding to the hexadecimal codes HEX(00) through HEX(7F). The second 128 byte-positions should represent conversions for incoming upshifted characters corresponding to the hexadecimal codes HEX(80) through HEX(FF).
3. If end-of-record character detection is enabled, the high-order bit for codes in the receive-code translation table must be set to 1 for each character defined as an end-of-record character (and set to zero for other characters).

Disconnect

```
$GIO DISCONNECT /01C (4406, G$)
```

The controller disconnects from the line by setting the Data Terminal Ready signal to zero for a period of three seconds.

Send Break

```
$GIO BREAK /01C (4407, G$)
```

The controller sends a break signal in accordance with the circuit and polarity denoted by the low-order hexdigit in byte-position-2 of the communications control vector. See Table 2-1 and Section 1.13.

Start Receiving Data

```
$GIO START RCV /01C (4408, G$)
```

One "start receiving data" statement is needed to enable data reception via the controller whether set for the full or half duplex mode. If set for half-duplex mode, the transmit and receive buffers are cleared first. The controller enters the receive mode and starts receiving data. Also, the receive timeout countdown is started by initiating byte 6 of the communications status vector to the value specified as the timeout interval (if different from binary zero).

Transfer Received Data to the CPU

```
$GIO RCV /01C (4409 C620, G$) D$()
```

or for a 2200MVP central processor

```
$GIO RCV /01C (4409 1020 02FF 03FF 1223 C620, G$) D$()
```

All or part (if an end-of-record character is detected) of the receive buffer characters are transferred from the controller to the CPU and stored in the array D\$(). (See Section 1.11.) The \$GIO data buffer, denoted here by D\$(), should be at least 255 bytes long since the controller has a 255-byte receive buffer. Bytes 9 and 10 in the error/status/general-purpose registers provided by the variable G\$, i.e., arg-2 of the \$GIO statement, are set to the binary representation of the number of bytes transferred whether stored or not. In a program for a 2200MVP central processor, insert a line containing a \$BREAK statement before each line containing a "transfer received data" statement.

Send Data

```
$GIO SEND /01C (440A A000 440C, G$) F$() <1, N>
```

If set for half-duplex mode, the receive buffer is cleared first. For half or full duplex mode, bytes 1 through N of the array F\$() are transferred from the CPU to the controller where they are stored in the transmit buffer after code and remains in the transmit mode (if set for the half-duplex operation).

NOTES:

1. The \$GIO microcommand A000 implements a particular signal sequence repeatedly (once per character until each character in the arg-3 data buffer is transferred from the CPU to the controller). Unless a 2200VP central processor is being used, the \$GIO syntax requires a single argument format for arg-3. Therefore, generally speaking, a \$PACK statement should be used to pack multi-argument data into a single argument prior to executing the "send data" \$GIO statement if the application requires a specially formatted buffer.
2. If desired, data can be transmitted via the controller using a PRINT, PRINTUSING, or MAT PRINT statement in conjunction with \$GIO statements by employing techniques such as those described in Section 2.6.

Send Then Receive Data

```
$GIO SEND RCV /01C (440B A000 440C, G$) F$() <E>
```

This statement is applicable only for the half-duplex mode of operation. Beginning with the Eth byte, all remaining bytes of the array F\$() are transferred from the CPU to the controller for storage in the transmit buffer after code translation is performed, if enabled. The controller transmits the data and then executes a "start receiving data" operation.

NOTES:

1. For half-duplex communications, the "send data" \$GIO operation should be used to send all but the last bytes of data. The "send, then receive data" \$GIO operation should be used to send the last bytes of data, and afterwards the "transfer received data to CPU" should be used. Use of the "send, then receive data" \$GIO operation automatically implements a line turnaround procedure, thereby ensuring the controller's readiness to receive data without loss.
2. For full-duplex communications, only the "send data" and "transfer received data to CPU" \$GIO operations are needed; the "send, then receive data" \$GIO operation should not be used since, in full duplex mode, the controller remains in transmit and receive mode simultaneously and line turnaround does not occur.

Stop Transmitting

\$GIO STOP SEND /O1C (440C, G\$)

This statement is applicable for the full-duplex mode of operation, in cases where the CPU must stop transmission temporarily because a control sequence is received without clearing the contents of the transmit buffer. Transmission commences when a "send data" \$GIO operation or a "continue transmitting" \$GIO operation is executed.

Continue Transmitting

\$GIO CONTINUE SEND /O1C (44QD, G\$)

This statement can be used to restart transmission if the transmit buffer contains data and transmission has been halted by a "stop transmitting" \$GIO operation.

2.5 AN EXAMPLE

The program listed in this section illustrates how a Wang system equipped with a Buffered Asynchronous Communications Controller can be programmed to emulate a Teletype terminal. The Wang keyboard corresponds to the Teletype keyboard and the CRT corresponds to the Teletype printer. Many REM statements are included in the program to highlight special features such as the following:

1. An asynchronous format with 7 data bits per character, even parity, and 1 stop bit is specified.
2. Rate = 300 baud (i.e., line speed is 300 bits per second).
3. Mode = half-duplex with automatic deletion of null characters.
4. Break signal transmission/detection is enabled with a 200ms transmission interval and a 120ms detection interval.
5. End-of-record detection is enabled. The carriage-return and DC1 (X-ON) characters are defined as terminators in the receive code translation table. Each carriage-return $(0D)_{16}$ is translated to $(8D)_{16}$ prior to storage in the receiver buffer. Each DC1 or X-ON character is translated to $(A0)_{16}$ prior to storage in the receive buffer. Upon transfer to the CPU, the high-order eighth bit of these end-of-record characters is changed from 1 to 0; hence, each $(8D)_{16}$ becomes $(0D)_{16}$ which is the ASCII code for a carriage-return, and $(A0)_{16}$ becomes $(20)_{16}$ which is the ASCII code for a space character.¹⁶
6. Characters received with a parity or framing error are replaced by the substitute character $(7F)_{16}$, the ASCII code for a DEL character. Via the receive code translation table, each $(7F)_{16}$ is converted to a null character, i.e., $(00)_{16}$.

If desired, the program can be keyed into the CPU and saved on disk or cassette to serve as a test program. However, keep in mind that the program incorporates special features and cannot be used unless the following conditions are satisfied:

1. If the address of the controller is not 01C, change the SELECT statement in the program accordingly. See line 110.
2. A suitable modem must be available and modems at both ends of the communications link must be similar. See Section 1.4.
3. The number of data bits per character, the number of stop bits, the type of parity, and the transmission rate must be matched at both ends of the communications link. If necessary, adjust the values in the communications control vector. See lines 300 through 390.
4. If attempting to communicate with a host computer, find out what sign-on procedure is required.

The Sample Program (Requires modification for 2200MVP central processors)

```

10 REM EXAMPLE --TTY EMULATION-- KYBD FOR INPUT, CRT FOR OUTPUT
20 DIM C2$(16)16, L$(255)1, K$1, X$(20)1, Z$(7)1
21 REM C2$() IS A 256-BYTE RECEIVE CODE TRANSLATION TABLE
22 REM L$() IS A 255-BYTE CPU RECEIVE DATA BUFFER
23 REM K$ IS A 1-BYTE CPU KEYBOARD INPUT BUFFER
24 REM X$() IS A 20-BYTE COMMUNICATIONS CONTROL VECTOR
25 REM Z$() IS A 7-BYTE CPU ARRAY FOR READING STATUS VECTOR
30 REM .....INITIALIZATION MODULE BEGINS
40 REM ..DEFINE $GIO MICROCOMMANDS TO OPERATE CONTROLLER
50 G0$=HEX(4402A000440C) :REM SET CONTROL VECTOR
60 G1$=HEX(4403C620) :REM READ STATUS VECTOR
70 G3$=HEX(4405A000440C) :REM LOAD RCV TRANSLATE TABLE
80 G6$=HEX(4408) :REM START RECEIVING DATA
90 G7$=HEX(4409C620) :REM TRANSFER RECEIVED DATA
100 G9$=HEX(440BA000440C) :REM SEND-THEN-RECEIVE DATA
110 SELECT #1 01C :REM SELECT 2227B AS #1
120 REM ..DEFINE RCV TRANSLATION TABLE
130 INIT(00)C2$() :REM CLEAR RCV TRANSLATION TABLE
140 C2$(1)=HEX(00010203040506070809000B0C8D0E0F) :REM 00-0F
150 C2$(2)=HEX(10A012131415161718191A1B1C1D1E1F) :REM 10-1F
160 C2$(3)=HEX(202122232425262728292A2B2C2D2E2F) :REM 20-2F
170 C2$(4)=HEX(303132333435363738393A3B3C3D3E3F) :REM 30-3F
180 C2$(5)=HEX(404142434445464748494A4B4C4D4E4F) :REM 40-4F
190 C2$(6)=HEX(505152535455565758595A5B5C5D5E5F) :REM 50-5F
200 C2$(7)=HEX(606162636465666768696A6B6C6D6E6F) :REM 60-6F
210 C2$(8)=HEX(707172737475767778797A7B7C7D7E00) :REM 70-7F
220 REM ..SPECIAL MEANINGS IN THE ABOVE TABLE ARE AS FOLLOWS
230 REM HEX 0D (CARRIAGE RETURN) SET AS TERMINATOR
240 REM HEX 11 (DC1, X-ON) SET AS TERMINATOR/SHOW AS SPACE
250 REM HEX 7F (DEL, RUBOUT) CONVERT TO NULL
255 REM HEX 80 THRU FF CONVERT TO NULL

```

(continued on next page)

```

260 REM ..DEFINE COMMUNICATIONS CONTROL VECTOR
280 INIT(00)X#() :REM INITIALIZE CCV TO BINARY ZERO
290 REM ..IF THE CCV IS NOT SET TO 00, TIME DELAYS MAY OCCUR.
300 X$(1)=HEX(17) :REM STOP BITS=1, BAUD RATE=300
310 X$(2)=HEX(11) :REM MODE=HALF-DUPLEX, BREAK ENABLED
320 X$(3)=HEX(21) :REM DATA BITS=7, PARITY=EVEN
330 X$(4)=HEX(7F) :REM ERROR SUBSTITUTE CHARACTER=DEL
350 X$(6)=HEX(01) :REM END OF RECORD DETECTION
380 X$(9)=HEX(14) :REM BREAK SEND INTERVAL=200 MS
390 X$(10)=HEX(0C) :REM BREAK DETECT INTERVAL=120 MS
400 REM ..
410 PRINT HEX(03),,"EMULATE TTY 300 BAUD" :REM CLEAR CRT
415 PRINT TAB(9);"KEYBOARD FOR INPUT--CRT FOR OUTPUT"
420 D,I=1:PRINT
430 $GIO SET CONTROLS #1 (G0$,A$)X#()
440 $GIO SET RCV TABLE #1(G3$,A$)C2#()
450 $GIO START RCV #1 (G6$,A$)
455 REM .....END OF INITIALIZATION MODULE
456 REM ..PROMPT OPERATOR
460 PRINT "*****BEGIN SIGN-ON PROCEDURE*****"
470 PRINT TAB(5);"NOTE---S.F. '15 IS PROGRAMMED TO SEND A BREAK
    SIGNAL."
480 GOTO 540
500 REM ....MAIN LOOP BEGINS
510 REM ..OUTPUT KEYED DATA TO 2227B
520 $GIO SEND DATA #1(G9$,A$)K$
530 REM ..KEYBOARD/T.C. TEST LOOP
540 $IF DN /001,770 :REM TEST KYBD READY
550 $GIO READ STATUS #1 (G1$,A$)Z#()
560 IF Z$(3)>HEX(00)THEN 730 :REM TEST FOR ERRORS
570 IF Z$(1)>HEX(00)THEN 740 :REM TEST FOR BREAK
580 IF Z$(4)=HEX(00)THEN 540 :REM TEST FOR COUNT ZERO
590 $GIO TRANSFER RCVD DATA #1 (G7$,A$) L#(<I>
600 A= VAL(STR(A$,10)) :REM A IS COUNT
610 IF A+D>64 THEN 650 :REM BRANCH IF A>=64
620 $GIO /005(A000,A$)L#(<I,A> :REM DATA TO CRT IF A<64
630 D=D+A :GOTO 690 :REM D IS OUTPUT POINTER
640 REM ..DISPLAY OVERRUN
650 B=65-D :REM B IS LINE LENGTH
660 $GIO/005(A000 400D 400A,A$)L#(<I,B>:REM DATA,CR,LF TO CRT
670 D=A-B
680 $GIO /005(A000,A$)L#(<I+B,D> :REM NEXT LINE TO CRT
690 I=I+A :A=I-1
700 IF L$(A)<>HEX(0D)THEN 540
710 PRINT
720 INIT(00)L#() :D,I=1 :GOTO 540
725 REM ..L#() IS CLEARED WHEN A C.R. IS RECEIVED
730 GOTO 550 :REM RCV ERROR DETECTED
740 PRINT "...BREAK RECEIVED":GOTO 540

```

```

750 REM ...KEYBOARD LOGIC FOLLOWS
770 SELECT PRINT 005:KEYIN K#,790,880 :REM ACCEPT KYBD INPLT
790 IF K#<HEX(20) THEN 830
800 PRINT K#; :REM DISPLAY K# ON CRT
810 D=D+1:IF D<65 THEN 520:PRINT :D=1:GOTO 520
820 PRINT :INIT(00)L#():D,I=1:GOTO 520
825 REM ..BRANCH AS FOLLOWS FOR A CODE HEX 08 THRU 0D
830 ON VAL(K#)-7 GOTO 850,800,840,520,520,820 :GOTO 520
840 PRINT K#;:GOTO 520
850 D=D-1:IF D>0 THEN 860:D=64:PRINT HEX(0C);
860 PRINT HEX(082008);:GOTO 520
870 REM ..BRANCH AS FOLLOWS FOR AN S.F. CODE HEX 07 THRU 0F
880 ON VAL(K#)-6 GOTO 840,850,800,840,520,520,520,520,890 :GO
TO 520
890 PRINT "....SEND BREAK":$GIO #1(4407,A#)
900 GOTO 540
910 REM ...END OF KEYBOARD LOGIC
920 REM ....END OF MAIN LOOP

```

(See notes on next page.)

NOTES:

1. Wang's Teletype Emulation Utilities software system has many features not illustrated in the example given in this section. With the Teletype Emulation Utilities, data transmission and reception can be controlled over point-to-point, dial-up communications links between Wang systems and host computer systems which support Teletype-like line protocols. From the viewpoint of a Wang system operating under control of a Teletype emulation program, the keyboard is always active as an input device for data transmission, and the CRT is always active as an output device for data reception. Additionally, and optionally, stored data can be transmitted from a disk or from a cassette, and received data can be output to a printer, disk, or cassette. The active I/O devices can be changed by the operator during program operation. During an initial phase of program operation, a parameter selection module lets the operator choose a set of communications options to achieve compatibility with a host computer system. For convenience, a set of default conditions can be accepted, if suitable for a particular communications link; otherwise, the parameter module (via prompts on the CRT) permits the operator to select the following:

- . the desired baud rate
- . the type of parity
- . the number of data bits per character
- . the number of stop bits

and to indicate how the host computer system normally reacts interactively in the following ways:

- . whether or not the host echoes each received character.
- . whether the host, upon receipt of a line of data denoted by a carriage return character, automatically supplies only a line feed character, or supplies a line feed character followed by one or more characters, or supplies no characters.

2. If interested in additional information regarding Wang-developed software systems which can be used with a Buffered Asynchronous Communications Controller, contact the Wang Sales Office in the area where a Wang system is being used.

2.6 TRANSMISSION VIA PRINT, PRINTUSING, OR MAT PRINT STATEMENTS

If desired, data transmission can be implemented via PRINT, PRINTUSING, or MAT PRINT statements by employing a special technique which effectively preserves the essential structure of the "send data" \$GIO operation described in Section 2.4 by breaking the operation into three phases, two of which must be replaced by \$GIO operations introduced in this section.

Consider the following statement:

```
100 $GIO SEND DATA /O1C (440A A000 440C, G$) A$()
```

- 440A Sends the code (OA)₁₆ to the controller via a CBS strobe to initiate data transmission.
- A000 Implements a prescribed sequence repeatedly (once for each character) until all data in the arg-3 buffer A\$() is transferred from the CPU to the controller.
- 440C Sends (OC)₁₆ to the controller via a CBS strobe to terminate data transmission.

Now, for convenience, let's define two operations not shown in Table 2-3;

<u>Operation</u>	<u>Microcommand Sequence</u>
Start send	440A
End send	440C

Using the new \$GIO operations, consider the following alternative programming sequence as a replacement for the "send data" \$GIO operation shown in line 100:

```
100 $GIO START SEND /O1C (440A, G$)
110 SELECT PRINT O1C (185)
120 PRINT X(); A$(); C$(5,12)
130 SELECT PRINT 005 (64)
140 $GIO END SEND /O1C (440C, G$)
```

In the sequence represented by lines 100 through 140, line 120 effectively replaces the microcommand A000 in the original line 100. A PRINTUSING or a MAT PRINT statement could be used in line 120.

The technique described in this section must not occur in the programming logic until after the communications control vector has been set. When specifying the communications control vector, it may be desirable to include provision for automatic delays following transmission of special characters such as a carriage return or tab character sent to a printer. See Section 2.2 and Table 2-1.

APPENDIX A
ASCII CODE SET

The System 2200 character set is defined by 8-bit codes of the form $b_8b_7b_6b_5b_4b_3b_2b_1$, where $b_8=0$ and the bits b_7 through b_1 correspond to the ASCII (American Standard Code for Information Interchange) character set which has 128 assignment positions, as shown in Table A-1.

Wang CRT's and printers use the ASCII code set, but some units may not display all the graphic characters shown in Table A-1. In some cases, substitute graphic characters may be displayed by a Wang peripheral. For details, refer to the manual which accompanies a particular peripheral.

Table A-1. ASCII Code*

Low order: 4 bits High order: 4 bits		hex digit		hex digit															
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F		
0000	0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI		
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15		
0001	1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US		
		16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31		
0010	2	Space	!	"	#	\$	%	&	(apos.)	()	*	+	(comma)	(dash)	(period)	/		
		32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47		
0011	3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?		
		48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63		
0100	4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O		
		64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79		
0101	5	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	(underline)		
		80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95		
0110	6	grave accent	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o		
		96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111		
0111	7	p	q	r	s	t	u	v	w	x	y	z	{		}	~	DEL		
		112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127		

*Numbers in the lower right corner of each box represent the decimal equivalent of the binary and the hexadecimal code for the character shown in the box, e.g., A = (41)₁₆ = (01000001)₂ = (65)₁₀.

LEGEND FOR ASCII CONTROL CHARACTERS			
NUL	Null	DLE	Data Link Escape
SOH	Start of Heading	DC1	Device Control 1
STX	Start of Text	DC2	Device Control 2
ETX	End of Text	DC3	Device Control 3
EOT	End of Transmission	DC4	Device Control 4
ENQ	Enquiry	NAK	Negative Acknowledge
ACK	Acknowledge	SYN	Synchronous Idle
BEL	Bell (audible or attention signal)	ETB	End of Transmission Block
BS	Backspace	CAN	Cancel
HT	Horizontal Tabulation (punched card skip)	EM	End of Medium
LF	Line Feed	SUB	Substitute
VT	Vertical Tabulation	ESC	Escape
FF	Form Feed	FS	File Separator
CR	Carriage Return	GS	Group Separator
SO	Shift Out	RS	Record Separator
SI	Shift In	US	Unit Separator
		DEL	Delete

APPENDIX B SPECIFICATIONS

Power Requirements

Supplied by the CPU.

Electrical Connection

A 25-pin RS-232-C, CCITT V.24 compatible female plug facilitates hookup of a modem.

Cable

A 12-foot (3.6m) cable, equipped with 25-pin RS-232-C compatible male connectors on each end, is supplied as an accessory.

Asynchronous Transmission Rates

50, 75, 100, 110, 134.5, 150, 200, 300, 600, 1200, 1800, 2400, 3600, 4800, 7200 or 9600 bits per second.

Character Format Options

Parity: odd, even, or no parity.
Number of Data Bits: 5,6,7 or 8.
Number of Stop Bits: 1, 1.5 or 2.

Communication Mode

Full or half duplex.

Compatible Modems

Bell 103 or 202 type, or equivalent.
Null modem, available from Wang, for direct communications link.

Standard Warranty Applies

INDEX

Acoustic couplers	5
Application program	2,9-12,14,15,19-21,23
ASCII	9,22,23,32
Assignments, pin	4
Asynchronous controller	1,8,14,20
Asynchronous equipment	1-3
Asynchronous transmission	6-8
Bell modem	5,13,33
Bit, parity	6-8
Bit, start	6-8
Bit, stop	6-8
Bits, data	3,6-8,11,14,16,17,33
Bits, shift status	10,11,22
Bits, status vector	19
Break signal	2,5,12,13,16,18,20,24
Buffer, receive	8-10,19,21
Buffer, transmit	8,10,19,21
Bytes, control vector	3,9-18
Bytes, status vector	19
Cable	1,3,4,33
CCITT	1,4
Character count	19
Character format	2,33
Character substitution	2,9,16,18
Clock, real time	19
Clock, selectable speed	2
Code, ASCII	9,22,23,32
Code translation	2,9-11,14,20-23
Communication mode	2,3,33
Connector	1,33
Continue transmitting	20,26
Control vector	2,3,9-12,14-16,20,21
Countdown, timeout	12
CPU	1,2,8,12,20,33
DAA	3,4
Data bits	3,6-8,11,14,16,17,33
Data buffering	1,2,8
Delays, automatic	2,9,16,18
Deletion, null characters	2,10,17
Detection, break signal	2,12,16-18,21
Detection, end-of-record	2,11,12,16,18,19,23
Detection, parity error	9,18,19,21
Detection, receive buffer overrun	9,19
Detection, received timeouts	2
Disconnect	20,24

EIA	1,4
End-of-record characters	2,11,12,16,18,19,23
End send	31
Error, framing	9,18,19,21
Error, parity	9,18,19,21
Error, receive buffer overrun	9,19
Example, programming	26-30
Fill characters	10
Format, character	2,33
Format, control vector	14,15
Format, status vector	19
Framing error	9,18,19,21
Full duplex	2,17,20,24-26,33
Half duplex	2,17,20,24,25,33
Initialization information	2,15
Initialization module	14
Insertion, shift character	2,10
Installation, controller	3
Installation, modem	3
Interfacing	3,4
Line speeds	2,5
Line turnaround	25
Logic levels	4,6,7
MAT PRINT	25,31
Microcode	1
Microcommands	20-26,31
Microprocessor	1,5
Mode, communication	2,3,33
Model 2227B	1
Modem	1,3-5,13
Monitoring	2,12
Null characters	2,10,17
Null modem	1,3,33
Option 62	1
Parity	3,6-8,16-18,33
Pin assignments	4
Polarity, break signal	13
Polarity, controller signals	4
PRINT	25,31
PRINTUSING	25,31
Program control	1
Programming techniques	14-31
Random-access-memory	2,19
Rates, transmission	2,5,8,16,17,33
Read-only-memory	1
Reception	6-8
Receive buffer	8-10,19,21
RS-232-C	1-5

Send break	20,24
Send data	20,24
Send then receive	20,25
Shift characters	2,10,11,15,16,18,22,23
Signals	2,5,7,8,13,19
Specifications, control vector	17,18
Specifications, controller	33
Start bit/element	6
Start receiving	20,24
Start send	31
Statements, \$GIO	1,2,9,12,14,15,19-26,31
Status vector	9,11-13,19-21
Stop bit/element	3,6-8,16,17,33
Stop transmitting	20,26
Substitution characters	2,9,16,18
Tables, code translation	2,9-11,14,20-23
Table lookup	10,23
Teletype emulation	26-30
Timeout countdown	12
Timeout interval	12,16,18,19
Transfer to CPU	12,19,20,24
Transmission delays	2,9,16,18
Transmission modes	2,16,17,33
Transmission rate	2,5,8,16,17,33
Vector, communications control	2,3,9-12,14-16,20,21
Vector, status	9,11-13,19-21
Voltage	4,6