TABLE OF CONTENTS

THIS PAGE LEFT INTENTIONALLY BLANK

INTRODUCTION

DMODEM Version. 1.8 is a communications software package that is used with the Digalog Series 2030 Functional Tester. This software allows the 2030 to communicate with another computer system through terminal emulation, XModem, or Kermit.

DMODEM is invoked by typing "DMODEM" at the OS/9 prompt. The main menu may be called after the DMODEM logo appears.

Prior to running DMODEM the serial port to be used should be configured as described in the Electrical Interface section.

THIS PAGE LEFT INTENTIONALLY BLANK

ELECTRICAL INTERFACE

The serial communication ports, T1 through T4, are at the rear of the equipment cabinet of the Series 2030 Functional Tester. These serial channels support either RS-232 or RS-422 interfaces.

Ports T1 through T4 are configured as Data Communication Equipment (DCE). The pin-outs for both RS-232 and RS-422 are shown below. Note that the communication ports do not support hardware handshaking (RTS/CTS). Data flow control may be accomplished with the use of software handshaking (XON/XOFF).

Connector Pin-Out

RS-232 CONNECTOR PIN-OUT

Pin Number	Function
1	Frame Ground
2	Receive Data
3	Transmit Data
6	+12V through 5.6kΩ current limit resistor
7	Signal Ground
8	+12V through 5.6kΩ current limit resistor

RS-422 CONNECTOR PIN-OUT

Pin Number	Function
1	Frame Ground
2	- Receive Data
3	- Transmit Data
6	+12V through 5.6kΩ current limit resistor
7	Signal Ground
8	+12V through 5.6kΩ current limit resistor
11	+ Receive Data
18	+ Transmit Data

NOTE: The connector supplied on the rear of the Digalog 2030 is a DB-25 female AMP# 745967-7.

The desired interface for each port may be selected via jumpers located on the G-96 I/O Controller PC board. The jumper settings may be found in Tables 1 and 2 and the locations in Figures 1 and 2, respectively.

Table 1: G-96 I/O Controller Assy. #0000-2268

PORT T1

JUMPER NUMBER	JP1	JP2	JP3	JP4	JP5
RS-232	*	-	-	-	-
RS-422	-	*	*	*	-

PORT T2

JUMPER NUMBER	JP6	JP7	JP8	JP9	JP10
RS-232	*	-	-	-	-
RS-422	-	*	*	*	-

PORT T3

JUMPER NUMBER	JP11	JP12	JP13	JP14	JP15
RS-232	*	-	-	-	-
RS-422	-	*	*	*	-

PORT T4

JUMPER NUMBER	JP16	JP17	JP18	JP19	JP20

RS-232	*	-	-	-	-
RS-422	-	*	*	*	-

* = shunt inserted

\- = shunt removed

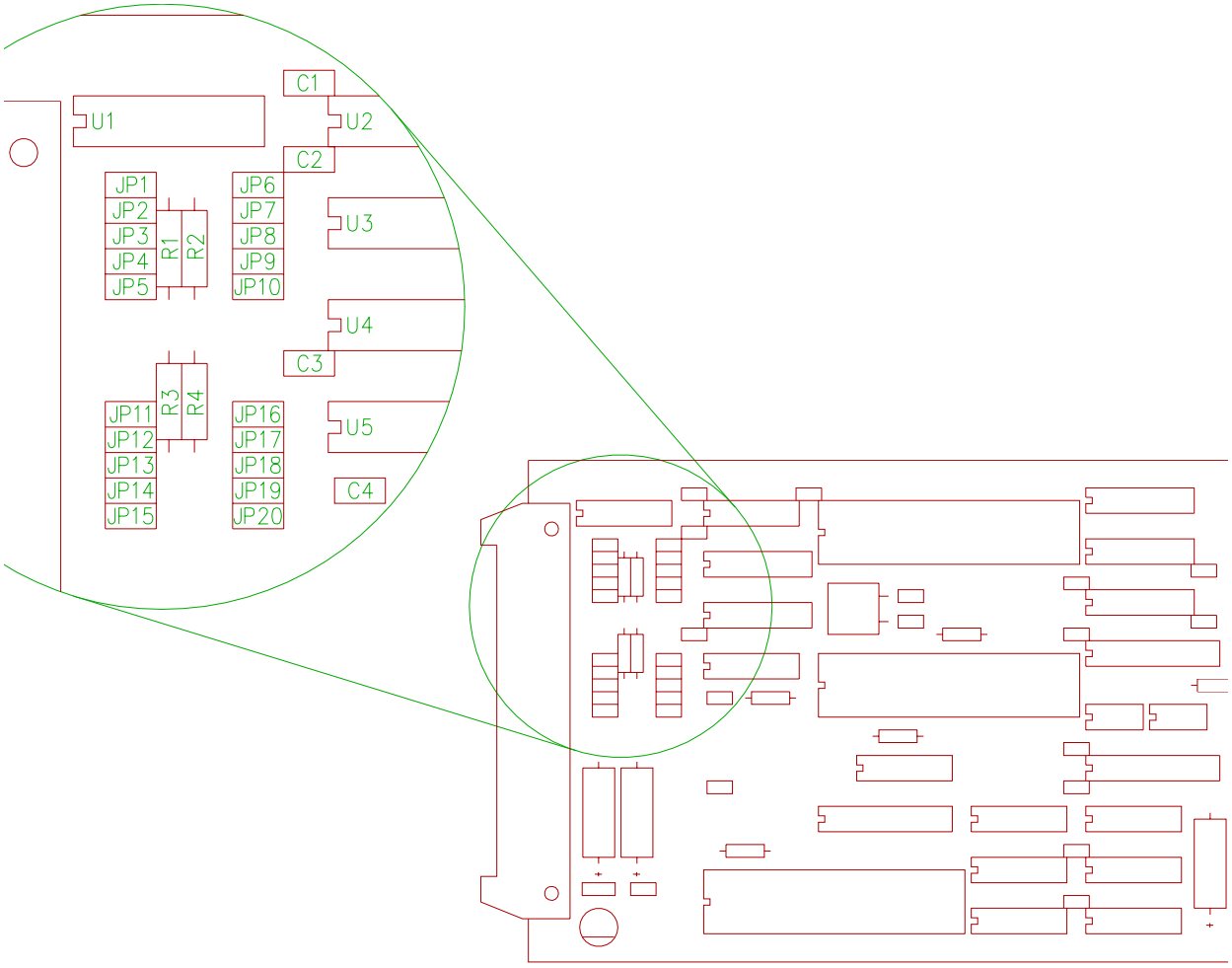

Figure 1: G-96 I/O Controller Assy. #0000-2268

Table 2: G-96 I/O Controller Assy. #0000-1568

PORT T1

JUMPER NUMBER	JP14	JP15	JP16	JP17	JP18
RS-232	*	-	-	-	-
RS-422	-	*	*	*	-

PORT T2

JUMPER NUMBER	JP19	JP20	JP21	JP22	JP23
RS-232	*	-	-	-	-
RS-422	-	*	*	*	-

PORT T3

JUMPER NUMBER	JP24	JP25	JP26	JP27	JP28
RS-232	*	-	-	-	-
RS-422	-	*	*	*	-

PORT T4

JUMPER NUMBER	JP29	JP30	JP31	JP32	JP33
RS-232	*	-	-	-	-

RS-422	-	*	*	*	-

* = shunt inserted

\- = shunt removed

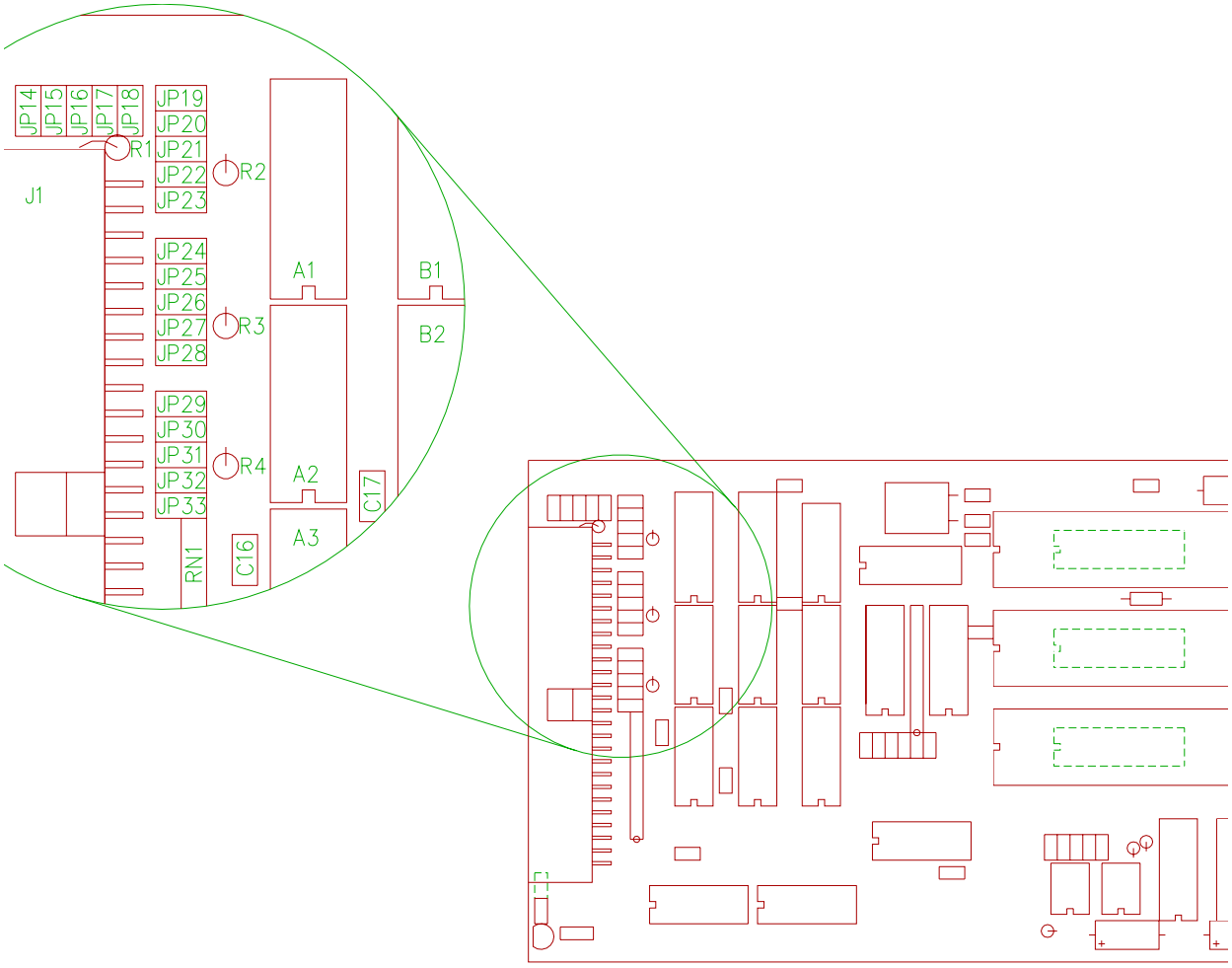

Figure 2: G-96 I/O Controller Assy. #0000-1568

THIS PAGE LEFT INTENTIONALLY BLANK

SOFTWARE INITIALIZATION AND SETUP

When DMODEM is initially executed it will generate a system initialization file called "profile". This file will contain all the default parameters required for user interaction. These parameters include the "Terminal Attention" character as well as the terminal X-ON and X-OFF characters. The system initialization file will also contain a default data transfer protocol of RAW DATA TRANSFER. This data transfer protocol is the first of six protocols available to the user. The following table lists all of the system defaults set when DMODEM is initially run.

Default Parameters

Parameter	Function
Attention Character.....	^I
Duplex.....	Half
Serial Port.....	/T3
Protocol.....	Raw text transfer
Baud Rate.....	9600
Terminal X-ON.....	^Q
Terminal X-OFF.....	^S
Host X-ON.....	^Q
Host X-OFF.....	^S
Block Size.....	128
Capture Unit Size.....	10
Format.....	8 bits
Stop bits.....	1
Parity.....	None

There are one of six data transmission protocols available to the user. The following table lists the protocols currently used.

Current Data Transmission Protocols

1. Raw Text Transfer
2. XMODEM Text Transfer
3. XMODEM Binary Transfer
4. Kermit Connect Mode
5. Kermit 7-bit Transfer
6. Kermit 8-bit Transfer

All of these protocols use 8-bit, no parity, and 1 stop bit format except for the Kermit 7-bit Mode which uses a format of 7-bits, no parity, and 2 stop bits. These parameters are set up automatically by DMODEM. ALL OF THE PARAMETERS MUST MATCH BETWEEN BOTH SYSTEMS IN ORDER FOR PROPER COMMUNICATION.

NOTE:The Capture Unit Size should be set to 100 for best operation of all the transmission

protocols.

SETTING A DATA TRANSMISSION PROTOCOL

Raw Text Transfer

The system initially defaults to the Raw Text Transfer protocol. To set the Raw Text Transfer the user must enter the Modify Options menu and select the Change Protocol (selection 6 in the Modify Options menu) to Raw Text Transfer (selection 1 in the change protocols).

Xmode Text Transfer

The Xmode corresponds to the Xmodem data transmission protocol. To set the Xmode Text Transfer the user must enter the Modify Options menu, select the Change Protocol (selection 6 in the Modify Options menu) and select the Xmode Text Transfer (selection 2 in the change protocols).

The best way to run this protocol is to set the error checking mode (selection 8 on the Modify Options menu) to CHECKSUM. The standard protocol for Xmodem is receiver driven meaning the machine receiving data should be started first. A 5 second delay should be allowed before transmitting from the sending machine. This allows the receiver to set the CHECKSUM mode.

Xmode Binary Transfer

The Xmode corresponds to the Xmodem data transmission protocol. To set the Xmode Binary Transfer the user must enter the Modify Options menu, select the Change Protocol (selection 6 in the Modify Options menu) and select the Xmode Binary Transfer (selection 3 in the change protocols).

The best way to run this protocol is to set the error checking mode (selection 8 on the Modify Options menu) to CHECKSUM. The standard protocol for Xmodem is receiver driven meaning the machine receiving data should be started first. A 5 second delay should be allowed before transmitting from the sending machine. This allows the receiver to set the CHECKSUM mode.

Kermit Connect Mode

The Kermit Connect Mode protocol is selected via the Change Protocol (selection 6) within the Modify Options menu by choosing Kermit Connect Mode (selection 4). It is advised to edit the initialization file, "profile", and change the duplex line from HALF to FULL. This is done because the half duplex mode will generate double characters on the terminal screen.

Kermit 7-bit Transfer

To set the Kermit 7-bit Transfer the user must enter the Modify Options menu and select the Change Protocol (selection 6 in the Modify Options menu) to Kermit 7-bit Transfer (selection 5 in the change protocols). With this mode the target machine must set its format to 7-bit, no parity, and 2 stop bits.

Kermit 8-bit Transfer

To set the Kermit 8-bit Transfer the user must enter the Modify Options menu and select the Change Protocol (selection 6 in the Modify Options menu) to Kermit 8-bit Transfer (selection 6 in the change protocols).

MAIN MENU

The Main Menu options are described below. Each option can be selected by entering the corresponding character or by using the cursor keys. At the top of the Main Menu DMODEM displays what port it is using, the baud rate which that port is transferring or receiving data, and whether the protocol is Kermit, Xmodem, or ASCII.

MODIFY OPTIONS

The Modify Options selection on the Main Menu should be chosen when first running DMODEM. This selection will give the user another menu of choices which are used to modify the current options.

Display Current Options

By entering "D" at the Modify Options Menu the computer will display the current options which are presently set. If the user desires to change these options most of them can be altered from the Modify Options Menu. To return to the Modify Options Menu press any key.

Change X-ON X-OFF Characters

This option is used to change the characters used to start and stop transmission in the raw text mode and xmodem text mode. Upon entering "1" at the Modify Options Menu a second menu will be displayed. The user then has the option of changing the external control characters, the internal control characters, or the tape mode control characters. These characters are entered as hexadecimal numbers.

Change EOL Mode

The options under this menu selection are used to change the way end of line (EOL) characters are handled for all types of protocol. To change the EOL mode, enter "2" at the Modify Options Menu.

The first option (ASIS) specifies that no translation of EOL characters will occur on input or output. This option should be used if straight binary transfer is desired.

The second option (CPM) specifies that a ‹NL› will be added after every ‹CR› on output. On input, the character sequence ‹CR› ‹NL› will be changed to a ‹CR›. If only a ‹NL› is present without a ‹CR› then the ‹NL› will be left as it is.

The third option (UNIX) specifies that each ‹CR› will be changed to a ‹NL› on output. On input, each ‹NL› will be changed to a ‹CR›.

The fourth option (SLOW) is used to add a delay after each output line. This allows a host, which has to do "housekeeping" between lines, to keep up.

The fifth option turns the slow mode off.

Change Capture Unit Size

This option is used to change the value of the capture unit size by entering a "3" at the Modify Options Menu. The unit size is used with the transfer block size to define the buffer size for the Xmodem mode. The buffer size is equal to the unit size multiplied by the transfer block size. The maximum unit size allowed is 100.

Change Attention Character

This option is used to define a character that when entered at the keyboard will cause a return to the Main Menu from the Terminal mode. To redefine the attention character, enter "4" at the Modify Options Menu and then enter the hexadecimal value of the desired character.

Change Transfer Block Length

This option changes the length of the Xmodem transfer block. To change this setting, enter "5" at the Modify Options Menu. The maximum allowable block length is 1280.

Change Protocol

There are six different types of protocols for transferring data which were discussed earlier in the Setting A Data Transmission Protocol section. To change the type of protocol, enter "6" at the Modify Options Menu and then select the number of the desired protocol. If the user enters a ‹CR› without selecting a new protocol, the program will remain at the type of protocol which is presently being used.

Toggle Error Check

DMODEM is capable of checking the data that is being transferred or received in two ways. These are CRC (Cyclic Redundancy Check) and CHECKSUM. By entering "8" at the Modify Options Menu the user can toggle between these two error checking options.

Toggle Debug Mode

The DMODEM program is written to allow the user to debug a problem in sending or receiving data. If there is a problem the user can toggle the debug mode ON, and the program will print to the screen what it is doing. This information can be used to find where the problems occur. If no problems are occurring, the debug mode can be left in the OFF position. The program will then run in a 'silent' mode which does not display the steps that the computer is taking. To toggle between the ON and OFF debug modes, enter "9" at the Modify Options Menu.

Change Baud Rate

To change the baud rate, enter "B" at the Modify Options Menu. The current baud rate will be displayed, and the user will be prompted to enter a new rate. If a ‹CR› is entered with no new rate, the baud rate will remain at its current value. To change the baud rate, enter the new value followed by a ‹CR›. Table 3, shown on the following page, contains the valid baud rates used by DMODEM.

Table 3: Valid baud rates used by DMODEM.

110
300
600
1200
2400
4800
9600
19200

Toggle LF/CR

This option is used with the Terminal mode to add a line feed after the carriage return if desired. This is commonly called "auto-line feed on carriage return". If this option is OFF when a ‹CR› is entered, a new line will not be started. If this option is ON when a ‹CR› is entered, a new line will be started. To toggle this option ON or OFF, enter "C" at the Modify Options Menu.

Change Port

As mentioned earlier, the port used defaults to T3. If the user would like to change the port used to a different serial port, this option can be used. To change serial ports, enter "P" at the Modify Options Menu. The computer will then ask the user to enter the new port. After the new port is entered, the user must also enter the desired baud rate for this port.

NOTE: If a ‹CR› is entered without designating a new port, the system will lock-up thus requiring the user to reboot the computer.

Quit And Return To Main Menu

To return to the Main Menu from the Modify Options Menu, enter "Q".

RECEIVE MODE

The Receive Mode is used to receive data in one of six different communication protocols. These protocols are selected

in the Modify Options Menu. In the raw text and Xmode protocols, the user has to define a storage file name. In the Kermit protocols, the storage file name is defined by the received data. The user can view the received raw text data by using the Main Menu View Buffer option or by reviewing the data file. In other data communication protocols, the user has to access the data file from the current directory.

The system checks for communication errors. When using Xmode and Kermit protocols, the system will abort reception after ten errors and exit to the Terminal Mode.

SEND MODE

The Send Mode is used to send data files with one of six different communication protocols. The communication protocol can be changed from the Modify Options Menu. The user has to designate which source file is to be sent for each type of protocol.

The system checks for communication errors. The system will abort transmission on the first error and exit to the Terminal Mode.

TERMINAL MODE

The Terminal Mode is used to configure the operation of the terminal as a dumb terminal for the remote system. The user can exit this mode by entering ‹Ctrl› {attention character}. Recall that the attention character can be changed in the Modify Options Menu.

DIAL PHONE NUMBER

The user can utilize the telephone link for data communication. This mode allows the user to enter a phone number to access the specific data communication link. The user can abort this mode by entering a ‹CR› and the system will exit to the Terminal Mode. If the phone number is not valid, the system will abort to the Terminal Mode.

HANG UP PHONE

This mode allows the user to disconnect the data communication link. The program will then go to the Main Menu.

VIEW BUFFER

This mode allows the user to view the received data in the buffer when using the raw text communication protocol.

FLUSH TEXT BUFFER

This mode clears the buffer contents and leaves the buffer open for further data reception under the current file name.

CLOSE BUFFER

This mode closes the current data buffer and saves its contents under the current file name.

KILL BUFFER

This mode clears the buffer contents including the current file name.

FORK A SHELL

This mode allows the user to access a shell. The user is prompted with ‹Dmodem› as a shell prompt. The user can switch back to the DMODEM Main Menu by entering ‹Ctrl› Z.

QUIT PROGRAM

This option is used to exit DMODEM.

THIS PAGE LEFT INTENTIONALLY BLANK

-------- 1. DEFINITIONS.

‹soh› 01H
‹eot› 04H
‹ack› 06H
‹nak› 15H
‹can› 18H
‹C› 43H

-------- 2. TRANSMISSION MEDIUM LEVEL PROTOCOL

Asynchronous, 8 data bits, no parity, one stop bit.

The protocol imposes no restrictions on the contents of the data being transmitted. No control characters are looked for in the 128-byte data messages. Absolutely any kind of data may be sent - binary, ASCII, etc. The protocol has not formally been adopted to a 7-bit environment for the transmission of ASCII-only (or unpacked-hex) data , although it could be simply by having both ends agree to AND the protocol-dependent data with 7F hex before validating it. I specifically am referring to the checksum, and the block numbers and their ones complement.

Those wishing to maintain compatibility of the CP/M file structure, i.e. to allow modemming ASCII files to or from CP/M systems should follow this data format:

* ASCII tabs used (09H); tabs set every 8.
* Lines terminated by CR/LF (0DH 0AH)
* End-of-file indicated by ^Z, 1AH. (one or more)
* Data is variable length, i.e. should be considered a continuous stream of data bytes, broken into 128-byte chunks purely for the purpose of transmission.
* A CP/M "peculiarity": If the data ends exactly on a 128-byte boundary, i.e. CR in 127, and LF in 128, a subsequent sector containing the ^Z EOF character(s)

is optional, but is preferred. Some utilities or
user programs still do not handle EOF without ^Zs.
* The last block sent is no different from others, i.e.
there is no "short block".

-------- 3. MESSAGE BLOCK LEVEL PROTOCOL

Each block of the transfer looks like:
‹SOH›‹blk #›‹255-blk #›‹--128 data bytes--›‹cksum›
in which:
‹SOH› =01 hex
‹blk #› = binary number, starts at 01 increments by 1, and wraps 0FFH to 00H (not to 01)
‹255-blk #› =blk # after going through 8080 "CMA" instr, i.e. each bit complemented in the 8-bit block number. Formally, this is the "ones complement".
‹cksum› =the sum of the data bytes only. Toss any carry.

-------- 4. FILE LEVEL PROTOCOL
---- 4A. COMMON TO BOTH SENDER AND RECEIVER:

All errors are retried 10 times. For versions running with an operator (i.e. NOT with XMODEM), a message is typed after 10 errors asking the operator whether to "retry or quit".

Some versions of the protocol use ‹can›, ASCII ^X, to
cancel transmission. This was never adopted as a standard, as having a single "abort" character makes the transmission susceptible to false termination due to an ‹ack› ‹nak› or ‹soh› being corrupted into a ‹can› and cancelling transmission.

The protocol may be considered "receiver driven", that is, the sender need not automatically re-transmit, although it does in the current implementations.

---- 4B. RECEIVE PROGRAM CONSIDERATIONS:

The receiver has a 10-second timeout. It sends a ‹nak› every time it times out. The receiver's first timeout, which sends a ‹nak›, signals the transmitter to start. Optionally, the receiver could send a ‹nak› immediately, in case the sender was ready. This would save the initial 10 second timeout.
However, the receiver MUST continue to timeout every 10 seconds in case the sender wasn't ready.

Once into a receiving a block, the receiver goes into a one-second timeout for each character and the checksum. If the receiver wishes to ‹nak› a block for any reason (invalid
header, timeout receiving data), it must wait for the line to clear. See "programming tips" for ideas

Synchronizing: If a valid block number is received, it will be: 1) the expected one, in which case everything is fine; or 2) a repeat of the previously received block. This should be considered OK, and only indicates that the receivers ‹ack› got glitched, and the sender re-transmitted; 3) any other block number indicates a fatal loss of synchronization, such as the rare case of the sender getting a line-glitch that looked like an ‹ack›. Abort the transmission, sending a ‹can›

---- 4C. SENDING PROGRAM CONSIDERATIONS.

While waiting for transmission to begin, the sender has only a single very long timeout, say one minute. In the

current protocol, the sender has a 10 second timeout before retrying. I suggest NOT doing this, and letting the protocol be completely receiver-driven. This will be compatible with existing programs.

When the sender has no more data, it sends an ‹eot›, and awaits an ‹ack›, resending the ‹eot› if it doesn't get one. Again, the protocol could be receiver-driven, with the sender only having the high-level 1-minute timeout to abort.

-------- 5. DATA FLOW EXAMPLE INCLUDING ERROR RECOVERY

Here is a sample of the data flow, sending a 3-block message. It includes the two most common line hits - a garbaged block, and an ‹ack› reply getting garbaged. ‹xx› represents the
checksum byte.

```
SENDER                                  RECEIVER
                                 times   out   after   10   seconds,
                          <---           <nak>
<soh> 01 FE -data- <xx> --->
                          <---           <ack>
<soh> 02 FD -data- xx   --->    (data gets line hit)
                          <---           <nak>
<soh> 02 FD -data- xx   --->
                          <---           <ack>
<soh> 03 FC -data- xx   --->
   (ack gets garbaged)    <---           <ack>
<soh> 03 FC -data- xx   --->           <ack>
<eot>                     --->
                          <---           <ack>
```

-------- 6. PROGRAMMING TIPS.

* The character-receive subroutine should be called with a
parameter specifying the number of seconds to wait. The
receiver should first call it with a time of 10, then ‹nak› and try again, 10 times.

After receiving the ‹soh›, the receiver should call the
character receive subroutine with a 1-second timeout, for the remainder of the message and the ‹cksum›. Since they are sent as a continuous stream, timing out of this implies a serious like glitch that caused, say, 127 characters to be seen instead of 128.

* When the receiver wishes to ‹nak›, it should call a "PURGE" subroutine, to wait for the line to clear. Recall the sender tosses any characters in its UART buffer immediately upon
completing sending a block, to ensure no glitches were mis-
interpreted.

The most common technique is for "PURGE" to call the
character receive subroutine, specifying a 1-second timeout, and looping back to PURGE until a timeout occurs. The ‹nak› is then sent, ensuring the other end will see it.

* You may wish to add code recommended by John Mahr to your character receive routine - to set an error flag if the UART shows framing error, or overrun. This will help catch a few more glitches - the most common of which is a hit in the high bits of the byte in two consecutive bytes. The ‹cksum› comes out OK since counting in 1-byte produces the

same result of adding 80H + 80H as with adding 00H + 00H.

-------- 7. OVERVIEW OF CRC OPTION

The CRC used in the Modem Protocol is an alternate form of block check which provides more robust error detection than the original checksum. Andrew S. Tanenbaum says in his book, Computer Networks, that the CRC-CCITT used by the Modem Protocol will detect all single and double bit errors, all errors with an odd number of bits, all burst errors of length 16 or less, 99.997% of 17-bit error bursts, and 99.998% of 18-bit and longer bursts.

The changes to the Modem Protocol to replace the checksum with the CRC are straight forward. If that were all that we did we would not be able to communicate between a program using the old checksum protocol and one using the new CRC protocol. An initial handshake was added to solve this problem. The handshake allows a receiving program with CRC capability to determine whether the sending program supports the CRC option, and to switch it to CRC mode if it does. This handshake is designed so that it will work properly with programs which implement only the original protocol. A description of this handshake is presented in section 10.

-------- 8. MESSAGE BLOCK LEVEL PROTOCOL, CRC MODE

Each block of the transfer in CRC mode looks like:
‹SOH›‹blk #›‹255-blk #›‹--128 data bytes-->‹CRC hi›‹CRC lo› in which:
‹SOH› =01 hex
‹blk #› =binary number, starts at 01 increments by 1, and wraps 0FFH to 00H (not to 01)
‹255-blk #› =ones complement of blk #.
‹CRC hi› =byte containing the 8 hi order coefficients of the CRC. ‹CRC lo›= byte containing the 8 lo order coefficients of the CRC. See the next section for CRC calculation.

-------- 9. CRC CALCULATION
---- 9A. FORMAL DEFINITION OF THE CRC CALCULATION

To calculate the 16 bit CRC the message bits are considered to be the coefficients of a polynomial. This message polynomial is first multiplied by X^16 and then divided by the generator polynomial (X^16 + X^12 + X^5 + 1) using modulo two arithmetic. The remainder left after the division is the desired CRC. Since a message block in the Modem Protocol is 128 bytes or 1024 bits, the message polynomial will be of order X^1023. The hi order bit of the first byte of the message block is the coefficient of X^1023 in the message polynomial. The lo order bit of the last byte of the message block is the coefficient of X^0 in the message polynomial.

---- 9B. EXAMPLE OF CRC CALCULATION WRITTEN IN C

```
/*
This function calculates the CRC used by the "Modem Protocol" The first argument is a pointer to the message block.
The second argument is the number of bytes in the message block. The message block used by the Modem Protocol
contains 128 bytes.
The function return value is an integer which contains the CRC. The lo order 16 bits of this integer are the coefficients
of the CRC. The lo order bit is the lo order coefficient of the CRC.
*/

int calcrc(ptr, count) char *ptr; int count; {

   int crc, i;

   crc = 0;
   while(--count >= 0) {
      crc = crc ^ (int)*ptr++ << 8;
      for(i = 0; i < 8; ++i)
         if(crc & 0x8000)
            crc = crc << 1 ^ 0x1021;
         else
            crc = crc << 1;
      }
   return (crc & 0xFFFF);
   }
```

-------- 10. FILE LEVEL PROTOCOL, CHANGES FOR COMPATIBILITY
---- 10A. COMMON TO BOTH SENDER AND RECEIVER:

The only change to the File Level Protocol for the CRC option is the initial handshake which is used to determine if both the sending and the receiving programs support the CRC mode. All Modem Programs should support the checksum mode for compatibility with older versions. A receiving program that wishes to receive in CRC mode implements the mode setting handshake by sending a ‹C› in place of the initial ‹nak›. If the sending program supports CRC mode it will recognize the ‹C› and will set itself into CRC mode, and respond by sending the first block as if a ‹nak› had been received. If the sending program does not support CRC mode it will not respond to the ‹C› at all. After the receiver has sent the ‹C› it will wait up to 3 seconds for the ‹soh› that starts the first block. If it receives a ‹soh› within 3 seconds it will assume the sender supports CRC mode and will proceed with the file exchange in CRC mode. If no ‹soh› is received within 3 seconds the receiver will switch to checksum mode, send a ‹nak›, and proceed in checksum mode. If the receiver wishes to use checksum mode it should send an initial ‹nak› and the sending program should respond to the ‹nak› as defined in the original Modem Protocol. After the mode has been set by the initial ‹C› or ‹nak› the protocol follows the original Modem Protocol and is identical whether the checksum or CRC is being used.

---- 10B. RECEIVE PROGRAM CONSIDERATIONS:

There are at least 4 things that can go wrong with the mode setting handshake.

1. the initial ‹C› can be garbled or lost.
2. the initial ‹soh› can be garbled.
3. the initial ‹C› can be changed to a ‹nak›.
4. the initial ‹nak› from a receiver which wants to receive in checksum can be changed to a ‹C›.

The first problem can be solved if the receiver sends a second ‹C› after it times out the first time. This process can be repeated several times. It must not be repeated a too many times before sending a ‹nak› and switching to checksum mode or a sending program without CRC support may time out and abort. Repeating the ‹C› will also fix the second problem if the sending program cooperates by responding as if a ‹nak› were received instead of ignoring the extra ‹C›.

It is possible to fix problems 3 and 4 but probably not worth the trouble since they will occur very infrequently. They could be fixed by switching modes in either the sending or the receiving program after a large number of successive ‹nak›s. This solution would risk other problems however.

---- 10C. SENDING PROGRAM CONSIDERATIONS.

The sending program should start in the checksum mode. This will insure compatibility with checksum only receiving programs. Anytime a ‹C› is received before the first ‹nak› or ‹ack› the sending program should set itself into CRC mode and respond as if a ‹nak› were received. The sender should respond to additional ‹C›s as if they were ‹nak›s until the first ‹ack› is received. This will assist the receiving program in determining the correct mode when the ‹soh› is lost or garbled. After the first ‹ack› is received the sending program should ignore ‹C›s.

-------- 11. DATA FLOW EXAMPLES WITH CRC OPTION
---- 11A. RECEIVER HAS CRC OPTION, SENDER DOESN'T

Here is a data flow example for the case where the receiver requests transmission in the CRC mode but the sender does not support the CRC option. This example also includes various transmission errors. ‹xx› represents the checksum byte.

```
SENDER                                  RECEIVER
                          <---          <C>
                                  times out after 3 seconds,
                          <---          <nak>
<soh> 01 FE -data- <xx> --->
                          <---          <ack>
<soh> 02 FD -data- <xx> --->    (data gets line hit)
                          <---          <nak>
<soh> 02 FD -data- <xx> --->
                          <---          <ack>
<soh> 03 FC -data- <xx> --->
   (ack gets garbaged)    <---          <ack>
                                  times   out   after   10   seconds,
                          <---          <nak>
<soh> 03 FC -data- <xx> --->
                          <---          <ack>
<eot>                     --->
                          <---          <ack>
```

---- 11B. RECEIVER AND SENDER BOTH HAVE CRC OPTION

Here is a data flow example for the case where the receiver requests transmission in the CRC mode and the sender supports the CRC option. This example also includes various transmission errors.
‹xxxx› represents the 2 CRC bytes.

```
SENDER                                    RECEIVER
                            <---          <C>
<soh> 01 FE -data- <xxxx> --->
                            <---          <ack>
<soh> 02 FD -data- <xxxx> --->    (data gets line hit)
                            <---          <nak>
<soh> 02 FD -data- <xxxx> --->
                            <---          <ack>
<soh> 03 FC -data- <xxxx> --->
   (ack gets garbaged)      <---          <ack>
                                    times   out   after   10   seconds,
                            <---          <nak>
<soh> 03 FC -data- <xxxx> --->
                            <---          <ack>
<eot>                       --->
                            <---          <ack>
```

THIS PAGE LEFT INTENTIONALLY BLANK

Digalog

DMODEM