2200 Terminal Controllers

Including the 2236 MXD, Triple controller, 2236 MXE and 2236 SVP

This document was written by Eric Wilson in August of 1983 at the tail end of the implementation of the MXE microcode revision R2.52.

MXE, SVP, MXD and Triple controller Specifications

The 2236 MXE and 2236 SVP controllers are the latest (and probably last) controllers in the series of 2200 MVP terminal multiplexer boards. Not withstanding the exceptions listed in the next paragraph the MXE and SVP are the same. Thus, whenever the MXE is discussed in this document the SVP is implied. The MXE provides all functions provided by the MXD and Triple Controller and has had many enahancements to functionality added. In the description which follows all four controllers are described together with all differences indicated. (the MXD should be treated as a subset of the Triple controller which is a supset of the MXE) All references to the MXD include the Triple controller.

The MXE and SVP are functionally equivalent but have some minor hardware differences. The SVP has only three RS232 ports while the MXE has four. Also, port 0 on the SVP does not supply all the RS232 signals needed for remote applications thereby limiting it to local terminals only. See the section on MXE/SVP RS232 pinout for full details.

This document defines the protocol between the 2200 and its various terminal controllers. Additionally, some other pertenent information about some controllers is included to help build a full picture. This document is intended as a supplement to the 2236MXE controller microcode.

The controllers can be enabled and accessed by a set of seven different device addresses. Typically 01h, 02h, 03h, 04h, 05h, 06h, and 07h. Since, however, the high order two bits of the controller address is switch settable, each controller can mapped to addresses 01h thru 07h, 41h thru 47h, 81h thru 87h, or C1h thru C7h.

Each device address is used for specific functions as stated in he following table:

Addresses	Function
01h and 05h	VP or bootstrap mode to 2236 terminal #1 only (power on, bootstrap and hardware errors, and running a normal VP).
02h	Receiving complete status of the terminal control (CRT, print buffer empty, entered lines ready, etc).
03h	MXE: Transfer Remote Screen Dump bytes from the controller to the 2200. MXD: Unused
04h	Sending a print line to the slave printer of the "currently selected" terminal. Also usable in VP mode for slave printer on terminal #1. MXE: Basic programs may talk to this address through $GIO commands.
06h	Controlling operations to and receiving data from the controller (selecting terminals, requesting line inputs, receiving line request data, etc.).
07h	Sending display data to the currently selected terminals's CRT.

OPERATION MODES

A. VP/Bootstrap Mode

When a 2200 CPU is first powered on, the bootstrap interacts with the terminal connected to the first port on the first controller. This is the MASTER terminal. No other terminal on the system may talk to the 2200 while in VP/Bootstrap mode (Exception: On an MXE the master terminal need not be the first terminal. Through the use of MXE command mode the MXE can be instructed to treat any one of terminals 1 through 4 as the master terminal. See MXE command mode description for full details).

(HISTORY: In this mode the master terminal acts like a 2226 terminal but does not support all possible ATOMS. The master terminal is the only terminal capable of setting the RESET and HALT/STEP strobes on the 2200 I/O bus.)

In VP/Bootstrap mode both controllers may be enabled at address 01h, 04h and 05h. Additionally, the MXE may be enabled at address 06h in BP/Bootstrap mode.

B. MVP Mode

MVP Mode is a special mode in which communication between all terminals and the 2200 is supported. In this mode, output may be sent to and input received from any of the 4 possible terminals. In this mode addresses 01h and 05h are not normally used. Enabling the controller at either of these two addresses causes the controller to return to VP/Bootstrap mode.

C. Set up of the MXE

In order to enter MVP mode on the MXE the controller must furst be downloaded with operating code. The 2200 does this in VP/Bootstrap mode then gives the MXE a command at address 06h to enter MVP mode. The MXE will then start executing the code loaded in RAM.

D. Setup of the MXD

Unlike the MXE, the MXD contains all necessary operating code within its 4 K Prom code space. When the 2200 enables the MXD at any of addresses 02h, 06h or 07h the MXD will begin operating in MVP mode (thus allowing all terminals to interact with the 2200).

Controller Buffer Usage:

Buffer	MXE	MXD	Use
CRT	250	250	To store characters to be transmitted to the terminal for displaying on its CRT.
PRINTER	250	160	To store characters to be transmitted to the terminal for printing on the slave printer.
LINE REQUEST	512	480	To store data associated with the line request.
KEYBOARD	36	36	To store all keystrokes received from the terminal. If a line request has been established then these are placed in the line request buffer. If no line request has been made these characters will be stored until: 1. A line request is made 2. A KEYIN command is issued 3. The buffer is full 4. The buffer is emptied via software command
Remote Screen Dump	256	none	To store Remote Screen Dump bytes from the terminal until the 2200 takes them. This function is not implemented in the MXD.

NOTE: The MXE uses 2 other buffers as follows:

A Receive buffer for temporary storage buffer of bytes taken from the DART.

A Data Transfer buffer for bytes to be transmitted to the terminal during Remote Screen Dump.

This list contains the commands from the 2200 to the controller:

HEX CODE	DESCRIPTION
FF	Select Terminal
00	Null
01	Poweron Sequence
02	Initialize Current Terminal
03	Delete Line Request
04	Keyboard Ready Check
05	Keyin Poll Request
06	Keyin Line Request
07	Line Request
08	Prefill Line Request
09	Refill Line Request
0A	End of Line Request
0B	Query Line Request
0C	Accept Line Request Data
0D	Request Crt Buffer
0E	Request Print Buffer
0F	Error Line Request
10	Terminate Line Request

All command from this point on are NOT implemented in the MXD. These commands will be ignored if issued to an MXD. Some of the commands are implemented in the Triple controller.

11*	Device status
12*	Start clock
13	Download code
14	End download
15	undefined
16	undefined
17	undefined
18	undefined
19	undefined
20	$MXE command
21	Return result of $MXE Command
22	Enable Connect
23	Disable Connect
24	Enable Disconnect
25	Disable Disconnect
26	Alert 2200 to ALL connected ports
27	Clear pending disconnection
28	Get ID of terminal
29	Initiate Remote Screen Dump
2A	Abort Remote Screen Dump
2B	Initiate Remote Screen Dump in responce to terminal request
2C	Reconnect port
2D	Select special package
2E	Set Controller password

* Commands implemented in the MXE and Triple controller but not the MXD.

Control

The 2200 passes commands to the controller at address 06h. The controller has no way to pass requests (i.e. commands from a slave device) to the 2200. The MXE may pass one byte commands to the 2200 at address 02h as explained in the section on the status address.

The following is a description of the command structure at address 06h:

Address 06h allows the MVP to define which terminal is to be communicated with and what tasks it is expected to perform. In particular, each command at address 06h is devided into primitive operations thereby allowing much flexibility. Some examples of command primitives are:

1. Cause the flow of data to be directed to and from a particular terminal (SELECT terminal).
2. Query the condition of a CRT or print buffer.
3. Define a line request.
4. Cause one or all terminals to be initialized.
5. Take single bytes from soft controller (KEYIN).
6. Query a line request completion.

The specific functions and command codes of addresses 06h are defined in more detail on the following pages. Although address 06h commands are described individually below, they can be transmitted to the controller in various combinations in a single stream.

The first byte of each command is sent with a CBS strobe. The subsequent bytes should be OBS's, sent without timeout. The controller should not be disabled during a command. Unless otherwise stated, output from the controller will be sent at address 06h.

NOTE: The examples given throughout the command section utilize the following symbols :

CBS(xx)	A Control Bus Strobe of byte xx is sent TO the controller.
OBS(xx)	An Output Bus Strobe of byte xx is sent TO the controller.
IBS(xx)	An Input Bus Strobe of byte xx is read FROM the controller.
ENDI(xx)	An Input Bus Strobe of byte xx with the 9th data bit SET is read FROM the controller.
CPB	The 2200 sets the CPU Busy signal active on the 2200 bus. This signals its readiness to receive bytes from the controller.

'xx' is a hexadecimal number.

For a more detailed description see the section on 2200 bus architecture.

SELECT TERMINAL CBS(FF) OBS(xx)

Whenever a command code of FF is received, the next byte will determine the terminal port which will be the subject of subsequent commands. All commands which are port specific (i.e. output to a terminal, input, etc.) act on the port most recently selected with this command. The data byte specifying the port must be a hexadecimal representation of the desired terminal (i.e., 00h = terminal #1, 01h = terminal #2, ..., 03h = terminal #4). For the remainder of this memo the current terminal is the last SELECTed terminal.

Example, select terminal #2:

CBS(FF) OBS(01 00)
(NOTE: The 2200 often ends a sequence with an OBS(00). This extra byte may or may not be needed for timing purposes)

POWER ON CBS(01)

MXE: The MXE jumps back into Prom and begins executing as a VP. All ports have been reset and the protocall between the 2200 and the controller must be the VP/Bootstrap protocall. The code downloaded to it previously is not harmed and may be restarted when the 2200 sends the proper command.

MXD The MXD reinitializes itself to VP/Bootstrap mode. Everything is set to the way it is at power on. All buffers are cleared, all pointers reset, all flags cleared. The mode becomes VP mode.

INITIALIZE CURRENT TERMINAL CBS(02)

This command will cause the CRT screen, pending line request CRT buffer, print buffer, and input buffer of the current terminal to be cleared (used at RESET).

Example,

CBS(02) OBS(00)

DELETE CURRENT LINE REQUEST CBS(03)

This command causes a pending line request and input buffers of the current terminal to be cleared (used at HALT and with special function keys).

Example,

CBS(03)

KEYBOARD READY CHECK CBS(04) IBS(xx)

Whenever a command code of 04h is received, the controller checks the keyboard buffer of the selected terminal. An IBS(00) is sent if it is empty, and non zero if there is a character (the non zero byte is of no particular value).

Example,

CBS(04) IBS(xx)

KEYIN POLL REQUEST CBS(05) IBS(xx)

Whenever a command code of 05h is received, the controller checks the keyboard buffer. If there is no character an IBS(00) is returned. Otherwise, a non zero byte is returned followed by an IBS of the byte (with ENDI if it is a Special Function key).

KEYIN LINE REQUEST CBS(06) IBS(xx)

Whenever a command code of 06h is received, the controller checks the keyboard buffer. If there is a byte in it, this command is treated exactly as the command 05h. If not, a IBS(00) is sent, and a special Keyin Line Request is set up. An interrupt (completed line request) will be generated at address 02h when a character is later received.

REQUEST LINE CBS(07) OBS(XXXXYYZZ)

A command code of 07h will cause the controller to setup to receive a field of up to XXXX characters (a hexadecimal representation of the count, not to exceed 480 (01E0h)) starting from the current CRT cursor position for the currently selected terminal. All field entries will be forced to stay within the field limits set. A line request is active until either a carriage return or a special function key is entered, or until a delete line request command is issued (RESET, HALT, etc). YY specifies three parameters as follows: The 80-bit specifies underline. The 04-bit specifies EDIT mode. The 01-bit specifies that characters previously entered in the keyboard buffer should be flushed. (In other words, keystrokes received prior to a line request being set, can be either received as part of the line or deleted). If deleted they are never echoed back to the CRT nor entered into the line request buffer. ZZ specifies current column of CRT cursor (the 2200 should have already positioned the cursor at this position).

Example,

CBS(07) OBS(00 C0 05 08)

sets up a 192 byte line request without underline starting in EDIT mode. The field begins in column 8.

PREFILL REQUEST LINE CBS(08) OBS(YYYY...)

This optional command code of CBS(08) can be sent after a line request command CBS(07) to prefill the desired line with the supplied characters YYY... starting with the leftmost position. The characters are treated as keystrokes. The cursor is left a the leftmost character. The string of characters is terminated by the next CBS, which will normally be an END-OF-LINE-REQUEST CBS(0A).

Example, prefill the line request with the string 'PREFILL':

assuming line request has been made

CBS(08) OBS(PREFILL)

Note: All characters are legal in a prefill, but when codes 00h-0Fh are being displayed at the terminal, they are changed to periods. Codes 80h-8Fh are also legal.

REFILL LINE REQUEST CBS(09) OBS(XXXX...)

The refill command is identical to a prefill except that it does not cause repositioning of the cursor to the beginning. Thus the characters are treated as keystrokes. It is normally used for RECALL AND DEFFN' quotes. It is generally followed by an END-OF-LINE-REQUEST CBS(0A) or a TERMINATE-LINE-REQUEST CBS(10).

END OF LINE REQUEST SEQUENCE CBS(0A)

A special command must be supplied to signal the end of a line request sequence which consists of the setup and prefill if desired. The last command sent, however, must be a CBS(0A), to signal the microcode to invoke the line request. Nothing is sent to the CRT until the CBS(0A) is issued.

Example,

assuming setup and prefill are complete.

CBS(0A)

This command is also used after usccessful RECALL or DEFFN' text entry to signal the controller to resume processing the line request.

NOTE: The 2200 will sometimes skip this command when it knows it will read in the data from the controller on the next command. This is a violation of protocol but the controllers live with it. Should the 2200 skip this command and go directly to CBS(0C) the controller should update the screen before allowing another Line Request to be started.

QUERY LINE REQUEST CBS(0B)

When an CBS(0B) command is received, the controller responds with one of the following IBS values.

```
      00h -- No line request in progress.
      01h -- line request still in progress.
      0Dh -- line request terminated by CR.
      FFh -- Recall key pressed  (see note).
ENDI(XX) -- S.F. key pressed.
```

Note on recall:

> After the FFh, the controller may send one or more bytes to the MVP. Each time the MVP sets CPB ready, the controller will send one more data byte with IBS. These are the characters from the entered text, read from right to left, beginning with the cursor position. The begnning of the buffer is indicated by ENDI. This sequence ends whenever the MVP stops setting CPB eady, sends OBS or CBS, or switches address. The controller should not clear the buffer when the 2200 has read all the bytes contained therein. Unfortunatly, the 2200 takes some shortcuts for expediency and may reread the buffer later.

Following the query, the MVP may do one of the following:

1. Nothing (another query later).

2. Delete line request (usually for HALT, and SF keys without parameters).

3. Refill -- this is more data to be merged to the present line request, as though the operator typed it (used for recall and DEFFN' quotes). Then End Line Request.

4. Terminate Line Request -- used to implement DEFFN' HEX(0D).

5. Error Line Request -- this beeps an error and continues the line request.

6. Ask for data.

ACCEPT LINE REQUEST DATA CBS(0C)

When a CBS(0C) is received after a line request has been completed, the controller will send the data. It should only be issued after a query has shown that the line is complete.

The controller sends the data if any, then an ENDI as terminator. if the ENDI is zero, the line request is complete; if 01h, the controller needs more time to finish updating the screen.

REQUEST CRT BUFFER CBS(0D)

This command causes the controller to check the CRT buffer of the current terminal. If it is empty, the appropriate status bit is set (address 02h = ready) to signal the fact. If not, then the controller will set the bit when the buffer does go empty.

REQUEST PRINT BUFFER CBS(0E)

This is just like the previous, except it refers to the current therminal's PRINT buffer, not CRT buffer.

ERROR LINE REQUEST CBS(0F)

This command causes the line request to resume, just like END-LINE-REQUEST, except it beeps first. It should not be used in conjunction with PREFILL or REFILL. It is normally used for undefined function keys.

TERMINATE LINE REQUEST CBS(10)

This command is used (after optional PREFILL or REFILL) to cause all the same actions as the operator pressing EXEC. It is normally used for the basic statement DEFFN' HEX(0D).

DEVICE STATUS CBS(11) IBS(xxxxxx) ENDI(00)

This command gets the controller type. Controllers which do not support this command will cause a timeout error in the 2200. When the timeout error is gotton the 2200 will assume that the controller is an MXD. The MXE and Triple controller support this command. The string returned by this command has the following format:

Byte 1	01	Triple controller without clock
	02	W/Time of Day clock
	04	W/Battery Backup TOD clock
	80	Needs Downloading (This bit may be set regardless of other bits)
Bytes 2 and 3		Code Revision:
Byte 2		Revision integer portion
Byte 3		Revision fractional portion times .01
Bytes 4 + ENDI(00)		Controller name

Start Clock CBS(12)

This command causes the board to begin issueing one second ticks to the 2200 at address 02h.

Download Code CBS(13) OBS(xxxx....)

This command signals the start of a block of RAM microcode for the controller. The 2200 does not care what the format of the block is. Each block will start with a CBS(13) and will be sent to the controller unbroken. It is assumed that the controller understands the format of the blocks.

End Download Code CBS(14)

This command signals the end of the downloading process. When the controller receives this command it will begin executing the downloaded code. Currently, the MXE is the only downloaded board. It expects to begin executing at the locations pointed at by location 4000h.

$MXE Command CBS(20) OBS(Command String)

This command has not been implemeted by the 2200 and is therefor not fully tested.
$MXE commands can be passed to the MXE preceeded by a CBS(20). Almost all the commands supported at the terminal end could be executed this way. The entire command string would be passed to the MXE without interpretation by the 2200.

Return Result of $MXE Command CBS(21)

This command would be used to get the result of an MXE command issued with CBS(13).

Enable Connect CBS(22)

This command enables the connect alert system for the currently selected port. This command sets a flag which the controller tests when a terminal connects to a device. If this bit is set (or the disconnect flag bit is set) then the controller alerts the 2200 to the connection through a connect flag at address 02h.
A connect is defined as the Data Set Ready signal on the RS232 line becoming active.

Disable Connect CBS(23)

This command disables the connect system for the currently selected port.

Enable Disconnect CBS(24) OBS(xxxx)

This command enables the disconnect alert system for the currently selected port. This command sets a flag which the controller tests when a terminal disconnects from a device. If this bit is set then the controller alerts the 2200 to the disconnection through a disconnect flag at address 02h.
The bytes xxxx define a timer value which is interpreted as follows:

> If xxxx = 0000 then the device is disconnected immediatly.
> If xxxx = FFFFh then any current disconnect countdown is canceled and the device is NOT disconnected by the controller.
> Otherwise, the value of xxxx is used a timer value and causes the controller to forcably disconnect the device after xxxx seconds.

NOTE: the controller disconnects the device by resetting DTR to it's inactive state until the 2200 allws reconnection by use of the command CBS(2C).

A disconnect is defined as the Data Set Ready signal on the RS232 line becoming inactive.
When a disconnect occurs AND the disconnect flag is set the controller should reset Data Terminal Ready to its inactive state for a period of 3 to 5 seconds so that any communication equipement connected to the port will let go of its communication line (such as a telephone line).

Disable Disconnect CBS(25)

This command disables the disconnect system for the currently selected port.

Alert 2200 to All Connected Ports CBS(26)

This command causes the controller to set the connect bit at address 02h for each port which is currently connected to an active device. This command does NOT set or reset any flags; it's action is one time only.

Clear Pending Disconnection CBS(27)

This command cancels a pending disconnection for the currently selected port but does not disable the disconnection detection system for that port.

Get ID of Terminal CBS(28) IBS(xxxx)

This command causes the controller to pass a 2 byte value to the 2200 which designates weather or not the currently connected device (at the currently selected port) supports RSD. The 2 bytes should be decoded as follows:

Byte 1: 00 = No RSD support
01 = Supports RSD

Byte 2: 00 = unknown terminal type
01 = 2336 DW terminal byte

Initiate Remote Screen Dump CBS(29) OBS(RSD parameters) ENDI(xx)

This command is used to begin a Remote Screen Dump from the currently selected port. If the device supports RSD then the controller returns a value of ENDI(01) to the 2200, initiates the RSD by sending the proper RSD start command to the device and passes the device the parameter string received from the 2200. If the device does not support RSD than the controller returns ENDI(00) to ther 2200 and discards the parameter string.

Parameters: N bytes to be sent to the terminal at the start of the screen dump which specify the area of the screen to be dumped.

xx: 00 = RSD cannot be done to this terminal.
01 = RSD begins

Abort Remote Screen Dump CBS(2A)

The 2200 issues this command to the controller when it wishes to stop a RSD already in progress. The Remote Screen Dump at the currently selected port is aborted and all RSD buffers are flushed. If no RSD is currently executing at the currently selected port the controller ignores this command.

Initiate Remote Screen Dump from Terminal Request CBS(2B)

This command is the same as CBS(29) except it is issued in responce to a RSD request issued by the device. The reason this is a seperate command is so that RSD requests can be queued up and the controller can keep track of them.

Reconnect Port CBS(2C)

This command instructs the controller to begin a 5 second timer which will then reconnect the currently selected device. This command is issued after the 2200 has completed all necessary disconnect procedures for a disconnecting device.

Select Special Package CBS(2D) OBS(xx) ENDI(yy)

This command selects a special support package such as Asynchronous Communications to be run at the currently selected port. The OBS byte is currently decoded as follows:

00h	NULL
01h	Standard Wang Terminal
02h	Asynchronous Communications Package (TC)

Set Controller Password CBS(2E) OBS(string)

The first six bytes of 'string' are set as the controller password.

Address 03h - Asynchronous Communications

Address 03h - Asynchronous Communications

Address 03h is only used by the MXE. It is used for communication between a Basic program tand the MXE Asynchronous Communications code. The user may use the $GIO command to perform input and output at this address useing the A## addressing convention. For a description of the protocol see the 'Asynchronous Communications Guide for 2236 MXE' Wang # 700-8089. Commands at this address may be broken at any time due to 2200 breakpointing. The user is not trusted as the 2200 is at other addresses and extensive error checking is done at this address. (at all other addresses the 2200 is TRUSTED not to send anything which will confuse the controller).

Address 07h - Send data to CRT

Address 07h is used to transmit characters onto the CRT of the current terminal. It should be ready whenever there is at least three bytes of space available. When it goes busy the CPU will time out (one millisecond) and service other partitions until a buffer-empty interrupt occurs. In addition to OBS with data to be displayed, this address also supports CBS's at this address. When the controller receives a CBS(xx) at address 07h it should place a FBh before it in the buffer. This will cause the terminal to accept this string as a command. The following are the three possible commands to the terminal:

1. Repeat

   FBh count character Where count is in the range 00h to 50h

   A CBS(04) to CBS(50), followed by an OBS of any byte except FB will have the effect of generating 4 to 80 characters of that byte. The code will be stored and transmitted in compressed form and thus will save time. It can be used to easily generate position cursor or tab functions. (CBS(00) to CBS(03) will work, but with a loss in efficiency).

   Example,

   OBS(01) CBS(05) OBS(0A)

   will place the string 01h FBh 05h 0Ah in the CRT buffer and will move the cursor to the beginning of row 5.

   *Note: The OBS must be sent without timeout.

2. Pause

   FBh delay where delay is in the range C0h to C9h

   A CBS(C0) to CBS(C9) will cause the terminal to pause for 0 to 1.5 seconds (one-sixth second increment). The controller will still continue to accept data until its buffer is full.

3. Character FBh

   The character FBh must be followed by the byte D0h so that the terminal will not interpret it as the start of a command. The controller places a D0h in the buffer immediatly after storing the FBh.

Address 04h - Send Line to Slave Printer

Address 04h is similar to address 07h except that the characters are directed to the printer of the current terminal. This may be used in VP mode, as well as MVP mode, but then only with terminal #1. The terminal does not support pause or repeat to the printer but all FBh must be followed by D0h as at address 07h.
MXE only: Remote Screen Dump bytes are transfered to the 2200 at this address. Whenever the 2200 sets CPB ready at address 04h the MXE will send RSD bytes to the 2200 followed by an ENDI(00) when the buffer is empty.

Address 02h - Receive Controller Status

Address 02h is used to report the stati of the various terminals to the 2200. When enabled by address 02h with CPB ready the controller will send from 7 to 11 bytes of data and 1 ENDI data byte to be used as a terminator for the input sequence.

Bytes	Explanation	
1	Low nibble:	RESET flags (1-bit per terminal)
	High nibble (MXE only):	Connect flags
2	Low nibble:	Halt/step flags (1-bit per terminal)
2	High nibble (MXE only):	Disconnect flags

Note: The above 2 bytes are organized so that the 01 & 10-bits represent terminal 1, 02 & 20-bits terminal 2, 04 & 40-bits represents terminal 3 and the 08 & 80-bits terminal 4.

3 One second clock tick as follows (MXE and Triple controller only):

If the 2200 has commanded this controller to tick of seconds then it will interrupt the 2200 at each one second tick and will pass the number of seconds which have elapsed since the 2200 last read the board status in this byte. The 80-bit must be set for the 2200 to consider this count legal. It is legal for the 2200 to not read the status for as long as it wants and the board must maintain the elapsed time in this byte. A board should only use set the 80-bit in this byte if the 2200 has previously issued command CBS(12).

4a	Terminal 1 status
4b (MXE)	Terminal 1 request
5a	Terminal 2 status
5b (MXE)	Terminal 2 request
6a	Terminal 3 status
6b (MXE)	Terminal 3 request
7a	Terminal 4 status
7b (MXE)	Terminal 4 request

Note: The above 8 bytes are organized as follows:

Byte a:
Bit
- 01 - Print buffer became empty (requested)
- 02 - CRT buffer became empty (requested)
- 04 - Keyboard buffer not empty or Line Request complete
- 08 - RSD buffer almost full
- 10 - unused
- 20 - unused
- 40 - unused
- 80 - Request byte (byte b) follows this byte

Byte b:
This byte is a one byte request to the 2200 for some function to be performed. It is only sent to the 2200 when the terminal has a request to make. The following requests are currently defined:

00h	Null
01h	Request Remote Screen Dump to be performed
02h	Cancel currently executing RSD

2200 Input/Output Bus

This section describes the signals on the 2200 I/O bus and their uses. Those signals which do not pertain to the functioning of the 2200-controller protocol have been omitted (such as ground). The list of bus signals is as follows:

RESET	Reset strobe - used by the controller in VP mode to reset the CPU
PRIME	Halt/Step strobe - used by the controller in VP mode to Halt or Step the CPU
Address bus	8 bit address bus controlled by the 2200
ABS	Address Bus Strobe
Input Bus	8 bit data bus from the controller to the 2200
ENDI	ENDI bit - 9^{th} data bit in the direction of from the controller to the 2200 only
IBS	Input Bus Strobe - from the controller to the 2200
Output Bus	8 bit data bus from the 2200 to the controller
OBS	Output Bus Strobe - from the 2200 to the controller for data bytes
CBS	Control Bus Strobe - from the 2200 to the controller for control bytes
CPB	CPU Busy - Signal (not a strobe) from the 2200 signaling to the controllers that the 2200 is ready to input bytes.
Ready/Busy	Ready-Busy signal to the 2200 from the controller

The 2200 selects a controller by setting the Address Bus to the controller address and strobing ABS. All controllers not selected should then not use the bus in any way so that the 2200 can communicate with the selected controller. The Ready/Busy line is set by the controller whenever the controller is selected. If more than one address is found on the same controller the R/B signal is set to the selected sub-address.

The Output Bus is used by the 2200 to output bytes to the controller using the OBS and CBS strobes. The controller can distinguish the two strobes apart thus allowing both data bytes and control bytes to be sent to the controller.

The Input Bus is used to transfer data bytes from the controller to the 2200 with the IBS strobe. A ninth data bit is used in this direction called the ENDI bit. This bit is used to indicate special conditions such as end of data transfer, etc. The ENDI bit is actually a flag in a register in the 2200 rather than a strobed data bit.

CPB is set by the 2200 to indicate its readiness to accept bytes from the controller. With a few exceptions the controller must respond within 15MS or a timeout error will occur. Once the CPB signal is set either bytes are transfered or the timeout condition is encountered.

The RESET strobe is used by the terminal controller in VP mode to reset the CPU. This signal should not be strobed in MVP mode or the system will crash.

The PRIME signal, also called HALT/STEP is strobed by the terminal controller to signal the deesire to Halt between instructions or step through the next signal. This signal is used in MVP mode only.

MXE/SVP Option W RS-232c Connection Pinout

Pin #	EIA	CCITT	Name	Signal Source
1	AA	101	Protective Ground	
2	BA	103	Transmit Data	DTE
3	BB	104	Receive Data	DCE
4	CA	105	Request to Send	DTE
5	CB	106	Clear to Send	DCE
6	CC	107	Data Set Ready	DCE
7	AB	102	Signal Ground	
8	CF	109	Received Line Signal Detector*	DCE
9			Unconnected	
10			Unconnected	
11			Unconnected	
12			Unconnected	
13			Unconnected	
14			Unconnected	
15			Unconnected	
16			Unconnected	
17			Unconnected	
18			Remote Analog Loopback*	DTE
19			Unconnected	
20	CD	108.2	Data Terminal Ready*	DTE
21			Remote Digital Loopback*	DTE
22	CE	125	Ring Indicator*	DCE
23	CH/CI	111/112	Data Signal Rate Selector*	DCE/DTE
24			Unconnected	
25			Unconnected	

* This pin is not connected on port 1 of an SVP Option W (Ports 2 and 3 utilize these signals)

Baud Rates:

The MXD and Triple controller baudrate switches are usually marked to indicate the desired settings. For older controllers it may be necessary to consult the 2236 terminal guide for setting.

Below is a list of baud rates available on the MXE and SVP controllers. Some of the rates can only be set via the $MXE baud rate command and will not have an entry in the second column. Others can be set by the hardware baud rate switches. Please see the discussion following the table which describes how default hardware baudrates are set. When setting baudrates via the $MXE command, the baud rate should be entered exactly as found in this table.

Rate	Hardware switch setting
50	
75	
100	
110	0
134.5	1
150	2
200	3
300	4
600	5
1200	6
undefined	7
2400	8
undefined	9
4800	A
undefined	B
9600	C
19200	D
undefined	E
see note	F

NOTE: Setting all the hardware switchs to X'FF' will cause the MXE to go into diagnostic burn in mode. In this state the MXE will continuously loop on its board diagnostics and will not talk to the 2200 or allow any user input. All board errors found will be displayed on the screen.

MXE Code Structure

Eric Wilson

22 August 1983

This document contains a basic description of the MXE code and its structure. The user would be advised to read this document alongside a copy of the MXE microcode.

System Files

MXE source files are arranged in groups according to their package name. Each file name is an eight character name whose root describes the type of support. The main routine is ALWAYS found in a file ending with '00' and ALWAYS includes the names of all the other files required for the assembly. The list below describes all current main routines and there package function. NOTE: The actual disk file name does not include the '.' but the assembler and editor require it.

NAME	Package description
EW.MXEP00	MXE bootstrap code
EW.SVPP00	SVP bootstrap code
EW.MXED00	General MXE code. Supports all MXD functions plus Remote Screen Dump and MXE command mode. Base to which other packages may be appended.
EW.MPAC00	Special version of EW.MXED00. Does everything the latter does plus special MPAC Remote Editor. Acts as general base in the same mannar the latter does.
EW.MXET00	MXE Asynchronous support package. NOT standalone. Must be combined with EW.MXED00 or EW.MPAC00 (ie a general base).

Code Assembly

To assemble any of the above code the general 2200 Z-80 assembler (written by Max Blomme) should be used with main file (listed above). This code was written on the 2200 and can to date only be assembled and edited on a 2200 system!! The Prom files for both the MXE and SVP are assembled standalone and do not require the use of a symbol file. Changing the prom code may necessitate changes to the ram code as some entry points into the proms are at assumed locations. The Base packages should always be assembled using a symbol file so that any package assembled as an addition to the base can have access to all base variables. All code must have access to the entire buffer and main storage area.

Each base package contains an assembly date, revision number and ID message (which can be displayed in MXE command mode) located in module EW.xxxxBA where 'xxxx' is the package root node. This message should be changed before assembling the code for release.

If the base is to be used without another package (such as Asynch) then the 'SELPACK' block in module EW.xxxxCB should be modified to respond negativly to the 2200's select package request. The code description in that module supplies all needed information to make the change.

After assembling the code the various parts should be combined into one file using the 'Combine Object File' program. The Base Code should be loaded into the file first and the secondary package(s) second. To date, only the TC (Asynch) package exists as a secondary package.

Lastly, the file should be named '@MXE0' and copied onto the system disk. The 2200 will load the contents of file '@MXE0' into each MXE controller or SVP controller in the system at poweron time (only at poweron).

No checksums are maintained in the ram code files. However, a checksum must be generated for the prom files and the diagnostic code must be combined with the prom code. Use the combine object code program to add the diagnostic code to the MXE/SVP prom code. Then use the "Checksum Calculator" to add a checksum byte to the file. The checksum will replace the last byte in the code.

Entry points and polling

The entry point at powerup is location x'0000'. This location points to the diagnostic code which performs various diagnostic test on the board before passing control to the MXE/SVP prom code through location x'0003'. An absolute jump should be in each of these locations. Two types of diagnostic errors can be found by the diagnostic code; Fatal and Crippling. If a fatal error occurs the diagnostic code will not pass control to the bootstrap prom code. A crippling error will allow control to be passed but and a status code will be passed in register C. In both cases the diagnostic code will not turn off the diagnostic LED on the board (which was turned on when the board was reset at poweron). The state of the LED will not be changed by the bootstrap prom code. However, a bug exists in the way the prom code passes control information to the ram code such that the ram code does not know which state the LED should be set at. Since the LED bit shares a register with some other bits needed at poweron by the ram code it will ALWAYS be turned off at ram initilization.

The structure of the bootstrap code is very similar to the RAM code. The bootstrap was written first and then expanded into the RAM code. Many changes have been made to the RAM code since it diverged from the prom. Many new functions have been added necessitating restructuring of much of the code on numerous occasions. But it will prove easy to understand the prom code when the RAM code is understood. For this reason I do not present a description of the prom code here. To work with the prom code one should read the rest of this document and ignore any reference to labels which do not exist in the prom code.

Various labels and their associated functions

In the following descriptions the use of underlined characters in names designates that this is a label FORM and varies the underlined part.

example: 'CONBLCK#' where # is 0 to 3 designates the labels:
CONBLCK0
CONBLCK1
CONBLCK2
CONBLCK3

'GENSTORE' Start of the shared data area. Up to 2k of RAM is reserved for shared variables, etc. Only 1k is currently used.

'CONBLCK#' # is the port number (0 to 3). This label designates the begining of the ports data and buffer area. The first page contains ALL the ports varaibles and pointers so that paging is not needed. The rest of the area (and part of the first page) is devided into buffers. This data area is often refered to as the ports 'control block'

'STACK' Starting location of the system stack.

'START.P' Initialization entry point into the bootstrap.

'MVPSTART' General storage area in RAM.

'CURNTUSR' Pointer to currently selected ports control block.

'SWAPLIST' A table which describes a mapping between the 2200 port numbers and the actual control block addresses. This is used because the user can redefine port one to be any port before entering RAM. Thus, it is possible to swap any two ports but only port one is currently able to become any other port.

In the following descriptions 'BUF' can be any of KEY, CRT, PRT, LR, RX, DT or RSD. Each is the name of a buffer and has various pionters and counters associated.

'BUFBEG' Pointer to the start of the buffer. Contains the page address and assumes the offset of zero except in the case of the keyboard buffer ('KEYBUF'). The offset from 'KEYBUF' 'BUFBEG' since the keyboard buffer is not a full page (described later).

'BUFRD' Buffer read pointer. Wraps around at page boundry as most buffers are curcular (exception is Line Request buffer which is 2 pages long and is not curcular). ALWAYS read and increment.

'BUFWR'	Buffer write pointer. Wraps around at page boundry as most buffers are curcular (exception is Line Request buffer which is 2 pages long and is not curcular). ALWAYS write and increment.
'BUFCOUNT'	Number of bytes in buffer. NOTE: When the read and write pointers are equal the buffer is empty or overflowed (same as empty as count goes to zero).
'BUFBEG'	See this label for buffer sizes and constants.
'2200DATA'	MXE register and device map
'NOP'	List of constants
'DIAGBOOT'	diagnostic code entry points
'MVPSTART'	pointer to start of RAM code. This is located at location 4000h.
'INTVEC'	Interrupt vector table
'START'	initlial entry point into RAM code
'MAIN'	Top of polling loop
'TABLOOK'	Table lookup routine which is JUMPPED to from many parts of the code
'SEL##'	# here designates a 2200 address (0 to 7 but not 3) which needs service.
'POLLBUF'	Top of buffer polling loop
'OBS##'	Routine to check for Output Bus Strobe at address #
'CBS##'	Routine to check for Control Bus Strobe at address #
'CPB##'	Routine to check for CPU Busy to go active at address #
'C06##'	Routine to respond to 2200 command of ##. (see command list following SEL06 in code)
'USRFIND', 'FINDUSR', 'FINDUSRP' and 'FINDPORT'	Translate masks and pointers into other masks and pointers which point at data areas or flags specific to each port.
'LREDIT'	Line Request editor
'TXSERVER'	Transmitter service code

'STATEROA'	Start of receiver service table
'RXIN'	Service routine for received bytes. Bytes in the RXBUF are translated and interpreted and stored in the KEYBUF. A prescan has already been done in the receiver int routine but is repeated here for certainty.
'CLOCKI'	Clock int service routine
'CLOCK'	Clock init routine
'ENTX' and 'DISTX'	Manipulate WAIT/READY status line on DART chips to effectivly enable and disable transmitter interrupts.
'TIMEOUT'	Sets up an event timer for an event which must be performed later
'PUTBUF' and 'GETBUF'	These routines manipulate the various buffers. A suffix may often be found on the end of these names which imply that they perform some function similar to the routine without the suffix. Care should be used in modifying these routines to check all routines with similar roots for reentrancy.
'LROUT'	Main routine for manipulating screen in Line Requests. Most updating of the screen during LR's is done through this routine. Timing is very critical in this section of the code.
'INIT'	Start of the board initialization code. Calling this routine will completly reinit the board.
'RSTBUF'	Buffer reset code.

Interrupts

The use of the various devices is described in this section.

INT PRIORITY:

CTC00
DART0A
DART0B
CTC11

CTC channels 0 and 1 (CTC00 and CTC01) on both chips are used as time bases for the DART receiver/transmitters. The baud rates are controlled through these channels.
CTC channels 2 and 3 on chip one (CTC02 and CTC03) form a cascade counter for a one second clock int. This int is used to provide the 2200 with a one second tick and also as a flag to test various conditions. Events which must be timed are linked to this clock (so long as the time span is greater than one second).
CTC channel 2 on chip two (CTC12) is used to produce the transmitter ready int. It was decided not to use the transmitter empty int from the DART chips directly because this would put some transmitters at higher priorities than some receivers which could result in lost data. Therefor, the WAIT/READY outputs of the DARTs are ORed together, clocked and used as input to this CTC channel which is in cout mode. Thus, by manipulating the WAIT/READY function ont the dart chips the transmitter int's may enabled and disabled (see 'ENTX' and DISTX').
CTC channel 3 on chip two (CTC13) is not used in general terminal support. The TC (Asynchronous) package uses it to provide a source of 10MS int's for event timing.

Interrupt and Service routine pointers

Each port has a set of pointers which point at interrupt service routines, polling loop service routines, and a one second interrupt service routine. Each pointer is a two byte address with the low byte first (Z-80 convention). All pointers may be changed by the code running for a port at any time but MUST point at a valid address or be equal to zero!!! The pointers are as follows:

TXSTRT	Transmitter service routine
RXSTRT	Receiver service routine
RXSTRTS	Status change service routine
RXSTRTP	Parity/Framing error service routine
BUSTAB22	2200 service table
SERVLOC	Poll loop service routine
TESTSEC	One second service routine (run once per second)

Some routines are JUMPED TO (not called) with the B register equal to the currently selected ports control block. see an example of the type of routine being written before writing one. All interrupt routines are responsible for restoring any registers which where saved BEFORE the routine was called and for performing a RETI to end the int. Receiver int routines may use the auxilary register set but MUST leave interrupts disabled while useing them. No other service routines should use the auxilary register set!
The TESTSEC pointer should point at a routine which performs actions needed only on an occasional basis and is CALLED once per second (and should be kept as short as possible!!). Interrupts should be left enabled as much as possible. All interrupt routines should be as short as possible to prevent data loss on the receivers (which could result from interrupts being disabled for to long a period).
Interrupts are sometimes disabled for a few instructions at a time to provide a locking mechanism around devices (especially when programming a device) and buffer pointers which are accessed in both main loop and interrupts. This has proven to be a problem in the past since the Z-80 has no semiphores.

From: Eric Wilson
Subject: MXE Data Transfer Facility
Date: 03/28/83 (revision of 06/15/82 memo)
Pages: 4 (including cover)

Your comments are welcome and should be directed to Eric Wilson at X7192, M/S 1389A in Lowell Tower II.

c.c.

Proposal: Not implemented

PAM : 0039E

	(1) T - MXE (normal state)	(2) T-MXE (DT)		(3) MXE-T (DT)
F0	Edit key		D0	
F1		Escape	D1	
F2		Start Data	D2	
F3	Request Data Transfer		D3	Start Data Transfer
F4		End Data Transfer	D4	Abort Data Transfer
F5			D5	
F6			D6	
F7			D7	
F8	CRT go	CRT go	D8	Send
F9	PRT go	PRT go	D9	
FA	CRT stop	CRT stop	DA	Wait
FB	PRT stop	PRT stop	DB	
FC	Transparency		DC	
FD	ENDI/ATOM		DD	
FE	Dead key		DE	
FF			DF	

During a Data Transfer (DT) the MXE will use the codes in column 3 above to control the flow of bytes from the connected device. Column 2 contains the allowable commands from the device to the MXE during a DT. All the commands in column 2 must be preceeded by escape byte 'F1'. 'F1' in the normal data stream must also be escaped. Thus, a data byte of 'F1' would be sent as 'F1''F1'. An end of DT command would be sent as 'F1''F4'. Commands from the MXE to the device must be escaped by escape byte 'FB' which is the normal MXE to Terminal escape byte. 'FB' in the data stream will be converted to 'FB''D0' to be compatible with older terminal formats.

Data flow is allowed in both directions (to and from the connected device) to allow for the transmission of error checking codes in the reverse direction.

NOTE: A device MUST issue an'E5' to the MXE immediatly following EVERY reset and restart. This is used by the MXE as an indication that the device supports Data Transfer. The MXE will NOT allow DT with a device which has not issued an 'E5' since the last reset or restart. (This is to prevent a DT from being attempted with a device which does not support it).

General Command Sequence:

In the following description 'T-MXE' means 'From the device to the MXE', 'MXE-2200' is 'From the MXE to the 2200', and so on.

Device initiation:

T-MXE: 'F3''ID'*
MXE-2200: Set DT bit at address 02.
MXE-2200: Pass 'ID' at address 06.

A request has now been made to the 2200 for the desired Data Transfer. The MXE will continue in its normal mode until the 2200 initiates the DT as follows. (in the case of a terminal requesting a Remote Screen Dump, the terminal will also remain in its normal state until the 2200 initiates the RSD. This is to prevent the MXE and terminal from hanging while waiting for the 2200 to get ready. The 2200 may not be able to honor the terminals request at all!):

2200-MXE: Command at address 06 - pass 'ID' followed by any needed parameters for the requested DT.

The 2200 is now ready to proceed with the transfer and will wait for the devices to proceed.

MXE-T: 'FB''F3''ID''parameters'

Both the device and the MXE prepare for the DT. When the MXE is ready:

MXE-T: 'FB''F8'

The device should now begin the transfer. As needed the MXE will send the following Go and Wait codes to the terminal to control the flow of data once the transfer has begun:

Go: 'FB''F8'
Wait: 'FB''FA'

Five conditions may terminate a DT. They are as follows:

1) Normal termination: T-MXE: 'F1''F4'
MXE-2200: Set DT bit at address 02
MXE-T: 'FB''F8'

2) 2200 Abort: 2200-MXE: Abort DT command at address 06
MXE-T: 'FB''F4'
MXE-T: 'FB''F8'

* See section on ID for a complete description

3)	Reset Abort:	T-MXE:	'F1''12'
		MXE-2200:	Set DT bit at address 02
		MXE-2200:	Set Reset bit at address 02
		MXE-T:	Normal terminal reset procedure
4)	Device disconnect:	MXE-2200:	Set DT bit at address 02
		MXE-2200:	Set Disconnect bit at address 02
5)	MXE Abort:	MXE-T:	'FB''F4'
		MXE-T:	'FB''F8'
		MXE-2200:	Set DT bit at address 02

It should be noted that the device connected to the MXE does not have to be a terminal. It could be some other device (EX: Wangwriter) which usually looks like a terminal to the MXE but may use the MXE for Data Transfer other than a Remote Screen Dump. For this reason, the above protocall has been kept as general as possible. The MXE never has to know the format of the data being transfered nor the number of bytes.

ID:

A one byte ID is used to designate which type of Data Transfer is being requested (by the device) and which type is being initiated (by the 2200). Currently, only two ID's exist. An ID of 01 specifies Remote Screen Dump and an ID of 02 specifies File Transfer. As additional uses for the Data Transfer facility of the MXE are generated, new IDs will be assigned. In any case, the MXE does not care which transfer is being done so the ID is not important to it at this time.

Following the ID, the MXE will send to the terminal a list of parameters (four bytes in the case of RSD). For RSD this parameter list is a specification of the part of the screen which is to be dumped. The first two bytes specify the upper left corner of the area while the second two bytes specify the lower right corner. The first byte of each group is the column number (from 0 to 79 for a 2336DW terminal) and the second byte is the row number (from 0 to 25 for a 2336DW terminal). The MXE receives the parameters from the 2200 when it issues a DT command.

Compression (RSD):

Data compression has been defined for an RSD. The byte 'EC' followed by a count of the number of bytes being compressed followed by the byte being compressed will be used. To send an 'EC' the terminal must send 'EC''EC'. Counts of 'EC' (decimal 236) and 'F1' (decimal 241) are not allowed (ie. a string of 236 or 241 compressible bytes must be broken into two compressions).

'EC' 'count' 'compressed byte'

To: Bruce Paterson

From: John S. Deutsch

CC: Eric Wilson

Date: September 1, 1983

Subject: MXE Code extension proposal

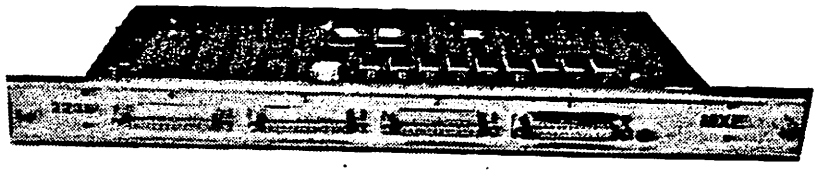
I have reviewed Eric's proposal to add extensions to the Async MXE code set. I believe that this is a very good idea. I know that you are are anxious to have Eric wrap up 2200 MXE activity, However I believe that this is a good investment. The added function in the Async code will help in competitive and development situations.

Your support for this project is appreciated and Eric's professional attitude towards the MXE project is to be commended. Let me know if you require any additional input in this area.

WANG

CUSTOMER ENGINEERING

PRODUCT MAINTENANCE MANUAL



2236 MXE TERMINAL CONTROLLER (EARLY FIELD SUPPORT)

MAY 1982

28 Nov 1983
Eric Wilson

These documents provide pieces of information describing the history of MxE development and past proposals. Much of the information in these docs is contained in previous sections of this notebook. Read the previous sections first.

These documents are contained on the diskette included with this notebook.

DOCUMENT SUMMARY

Document Id: 0015E
Document Name: MXE Specifications
Operator: Eric
Author: Eric

Comments:

Pages to be printed 21

Notify U13 on system PAM.

2200 Terminal Controllers

Including the 2236 MXD, Triple controller, 2236 MXE and 2236 SVP

This document was written by Eric Wilson in August of 1983 at the tail end of the implementation of the MXE microcode revision R2.52.

The 2236 MXE and 2236 SVP controllers are the latest (and probably last) controllers in the series of 2200 MVP terminal multiplexer boards. Not withstanding the exceptions listed in the next paragraph the MXE and SVP are the same. Thus, whenever the MXE is discussed in this document the SVP is implied. The MXE provides all functions provided by the MXD and Triple Controller and has had many enahancements to functionality added. In the description which follows all four controllers are described together with all differences indicated. (the MXD should be treated as a subset of the Triple controller which is a supset of the MXE) All references to the MXD include the Triple controller.

The MXE and SVP are functionally equivalent but have some minor hardware differences. The SVP has only three RS232 ports while the MXE has four. Also, port 0 on the SVP does not supply all the RS232 signals needed for remote applications thereby limiting it to local terminals only. See the section on MXE/SVP RS232 pinout for full details.

This document defines the protocol between the 2200 and its various terminal controllers. Additionally, some other pertenent information about some controllers is included to help build a full picture. This document is intended as a supplement to the 2236MXE controller microcode.

The controllers can be enabled and accessed by a set of seven different device addresses. Typically 01h, 02h, 03h, 04h, 05h, 06h, and 07h. Since, however, the high order two bits of the controller address is switch settable, each controller can mapped to addresses 01h thru 07h, 41h thru 47h, 81h thru 87h, or C1h thru C7h.

Each device address is used for specific functions as stated in he following table:

Addresses	Function
01h and 05h	VP or bootstrap mode to 2236 terminal #1 only (power on, bootstrap and hardware errors, and running a normal VP).
02h	Receiving complete status of the terminal control (CRT, print buffer empty, entered lines ready, etc).
03h	MXE: Transfer Remote Screen Dump bytes from the controller to the 2200. MXD: Unused
04h	Sending a print line to the slave printer of the "currently selected" terminal. Also usable in VP mode for slave printer on terminal #1. MXE: Basic programs may talk to this address through $GIO commands.
06h	Controlling operations to and receiving data from the controller (selecting terminals, requesting line inputs, receiving line request data, etc.).
07h	Sending display data to the currently selected terminals's CRT.

OPERATION MODES

A. VP/Bootstrap Mode

When a 2200 CPU is first powered on, the bootstrap interacts with the terminal connected to the first port on the first controller. This is the MASTER terminal. No other terminal on the system may talk to the 2200 while in VP/Bootstrap mode (Exception: On an MXE the master terminal need not be the first terminal. Through the use of MXE command mode the MXE can be instructed to treat any one of terminals 1 through 4 as the master terminal. See MXE command mode description for full details).

(HISTORY: In this mode the master terminal acts like a 2226 terminal but does not support all possible ATOMS. The master terminal is the only terminal capable of setting the RESET and HALT/STEP strobes on the 2200 I/O bus.)

In VP/Bootstrap mode both controllers may be enabled at address 01h, 04h and 05h. Additionally, the MXE may be enabled at address 06h in BP/Bootstrap mode.

B. MVP Mode

MVP Mode is a special mode in which communication between all terminals and the 2200 is supported. In this mode, output may be sent to and input received from any of the 4 possible terminals. In this mode addresses 01h and 05h are not normally used. Enabling the controller at either of these two addresses causes the controller to return to VP/Bootstrap mode.

C. Set up of the MXE

In order to enter MVP mode on the MXE the controller must furst be downloaded with operating code. The 2200 does this in VP/Bootstrap mode then gives the MXE a command at address 06h to enter MVP mode. The MXE will then start executing the code loaded in RAM.

D. Setup of the MXD

Unlike the MXE, the MXD contains all necessary operating code within its 4 K Prom code space. When the 2200 enables the MXD at any of addresses 02h, 06h or 07h the MXD will begin operating in MVP mode (thus allowing all terminals to interact with the 2200).

Controller Buffer Usage:

Buffer	MXE	MXD	Use
CRT	250	250	To store characters to be transmitted to the terminal for displaying on its CRT.
PRINTER	250	160	To store characters to be transmitted to the terminal for printing on the slave printer.
LINE REQUEST	512	480	To store data associated with the line request.
KEYBOARD	36	36	To store all keystrokes received from the terminal. If a line request has been established then these are placed in the line request buffer. If no line request has been made these characters will be stored until: 1. A line request is made 2. A KEYIN command is issued 3. The buffer is full 4. The buffer is emptied via software command
Remote Screen Dump	256	none	To store Remote Screen Dump bytes from the terminal until the 2200 takes them. This function is not implemented in the MXD.

NOTE: The MXE uses 2 other buffers as follows:

A Receive buffer for temporary storage buffer of bytes taken from the DART.

A Data Transfer buffer for bytes to be transmitted to the terminal during Remote Screen Dump.

This list contains the commands from the 2200 to the controller:

HEX CODE	DESCRIPTION
FF	Select Terminal
00	Null
01	Poweron Sequence
02	Initialize Current Terminal
03	Delete Line Request
04	Keyboard Ready Check
05	Keyin Poll Request
06	Keyin Line Request
07	Line Request
08	Prefill Line Request
09	Refill Line Request
0A	End of Line Request
0B	Query Line Request
0C	Accept Line Request Data
0D	Request Crt Buffer
0E	Request Print Buffer
0F	Error Line Request
10	Terminate Line Request

All command from this point on are NOT implemented in the MXD. These commands will be ignored if issued to an MXD. Some of the commands are implemented in the Triple controller.

11*	Device status
12*	Start clock
13	Download code
14	End download
15	undefined
16	undefined
17	undefined
18	undefined
19	undefined
20	$MXE command
21	Return result of $MXE Command
22	Enable Connect
23	Disable Connect
24	Enable Disconnect
25	Disable Disconnect
26	Alert 2200 to ALL connected ports
27	Clear pending disconnection
28	Get ID of terminal
29	Initiate Remote Screen Dump
2A	Abort Remote Screen Dump
2B	Initiate Remote Screen Dump in responce to terminal request
2C	Reconnect port
2D	Select special package
2E	Set Controller password

* Commands implemented in the MXE and Triple controller but not the MXD.

Control

The 2200 passes commands to the controller at address 06h. The controller has no way to pass requests (i.e. commands from a slave device) to the 2200. The MXE may pass one byte commands to the 2200 at address 02h as explained in the section on the status address.

The following is a description of the command structure at address 06h:

Address 06h allows the MVP to define which terminal is to be communicated with and what tasks it is expected to perform. In particular, each command at address 06h is devided into primitive operations thereby allowing much flexibility. Some examples of command primitives are:

1. Cause the flow of data to be directed to and from a particular terminal (SELECT terminal).
2. Query the condition of a CRT or print buffer.
3. Define a line request.
4. Cause one or all terminals to be initialized.
5. Take single bytes from soft controller (KEYIN).
6. Query a line request completion.

The specific functions and command codes of addresses 06h are defined in more detail on the following pages. Although address 06h commands are described individually below, they can be transmitted to the controller in various combinations in a single stream.

The first byte of each command is sent with a CBS strobe. The subsequent bytes should be OBS's, sent without timeout. The controller should not be disabled during a command. Unless otherwise stated, output from the controller will be sent at address 06h.

NOTE: The examples given throughout the command section utilize the following symbols :

CBS(xx)	A Control Bus Strobe of byte xx is sent TO the controller.
OBS(xx)	An Output Bus Strobe of byte xx is sent TO the controller.
IBS(xx)	An Input Bus Strobe of byte xx is read FROM the controller.
ENDI(xx)	An Input Bus Strobe of byte xx with the 9^{th} data bit SET is read FROM the controller.
CPB	The 2200 sets the CPU Busy signal active on the 2200 bus. This signals its readiness to receive bytes from the controller.

'xx' is a hexadecimal number.

For a more detailed description see the section on 2200 bus architecture.

SELECT TERMINAL CBS(FF) OBS(xx)

Whenever a command code of FF is received, the next byte will determine the terminal port which will be the subject of subsequent commands. All commands which are port specific (i.e. output to a terminal, input, etc.) act on the port most recently selected with this command. The data byte specifying the port must be a hexadecimal representation of the desired terminal (i.e., 00h = terminal #1, 01h = terminal #2, ..., 03h = terminal #4). For the remainder of this memo the current terminal is the last SELECTed terminal.

Example, select terminal #2:

CBS(FF) OBS(01 00)
(NOTE: The 2200 often ends a sequence with an OBS(00). This extra byte may or may not be needed for timing purposes)

POWER ON CBS(01)

MXE: The MXE jumps back into Prom and begins executing as a VP. All ports have been reset and the protocall between the 2200 and the controller must be the VP/Bootstrap protocall. The code downloaded to it previously is not harmed and may be restarted when the 2200 sends the proper command.

MXD The MXD reinitializes itself to VP/Bootstrap mode. Everything is set to the way it is at power on. All buffers are cleared, all pointers reset, all flags cleared. The mode becomes VP mode.

INITIALIZE CURRENT TERMINAL CBS(02)

This command will cause the CRT screen, pending line request CRT buffer, print buffer, and input buffer of the current terminal to be cleared (used at RESET).

Example,

CBS(02) OBS(00)

DELETE CURRENT LINE REQUEST CBS(03)

This command causes a pending line request and input buffers of the current terminal to be cleared (used at HALT and with special function keys).

Example,

CBS(03)

KEYBOARD READY CHECK CBS(04) IBS(xx)

Whenever a command code of 04h is received, the controller checks the keyboard buffer of the selected terminal. An IBS(00) is sent if it is empty, and non zero if there is a character (the non zero byte is of no particular value).

Example,

CBS(04) IBS(xx)

KEYIN POLL REQUEST CBS(05) IBS(xx)

Whenever a command code of 05h is received, the controller checks the keyboard buffer. If there is no character an IBS(00) is returned. Otherwise, a non zero byte is returned followed by an IBS of the byte (with ENDI if it is a Special Function key).

KEYIN LINE REQUEST CBS(06) IBS(xx)

Whenever a command code of 06h is received, the controller checks the keyboard buffer. If there is a byte in it, this command is treated exactly as the command 05h. If not, a IBS(00) is sent, and a special Keyin Line Request is set up. An interrupt (completed line request) will be generated at address 02h when a character is later received.

REQUEST LINE CBS(07) OBS(XXXXYYZZ)

A command code of 07h will cause the controller to setup to receive a field of up to XXXX characters (a hexadecimal representation of the count, not to exceed 480 (01E0h)) starting from the current CRT cursor position for the currently selected terminal. All field entries will be forced to stay within the field limits set. A line request is active until either a carriage return or a special function key is entered, or until a delete line request command is issued (RESET, HALT, etc). YY specifies three parameters as follows: The 80-bit specifies underline. The 04-bit specifies EDIT mode. The 01-bit specifies that characters previously entered in the keyboard buffer should be flushed. (In other words, keystrokes received prior to a line request being set, can be either received as part of the line or deleted). If deleted they are never echoed back to the CRT nor entered into the line request buffer. ZZ specifies current column of CRT cursor (the 2200 should have already positioned the cursor at this position).

Example,

CBS(07) OBS(00 C0 05 08)

sets up a 192 byte line request without underline starting in EDIT mode. The field begins in column 8.

PREFILL REQUEST LINE CBS(08) OBS(YYYY...)

This optional command code of CBS(08) can be sent after a line request command CBS(07) to prefill the desired line with the supplied characters YYY... starting with the leftmost position. The characters are treated as keystrokes. The cursor is left a the leftmost character. The string of characters is terminated by the next CBS, which will normally be an END-OF-LINE-REQUEST CBS(0A).

Example, prefill the line request with the string 'PREFILL':

assuming line request has been made

CBS(08) OBS(PREFILL)

Note: All characters are legal in a prefill, but when codes 00h-0Fh are being displayed at the terminal, they are changed to periods. Codes 80h-8Fh are also legal.

REFILL LINE REQUEST CBS(09) OBS(XXXX...)

The refill command is identical to a prefill except that it does not cause repositioning of the cursor to the beginning. Thus the characters are treated as keystrokes. It is normally used for RECALL AND DEFFN' quotes. It is generally followed by an END-OF-LINE-REQUEST CBS(0A) or a TERMINATE-LINE-REQUEST CBS(10).

END OF LINE REQUEST SEQUENCE CBS(0A)

A special command must be supplied to signal the end of a line request sequence which consists of the setup and prefill if desired. The last command sent, however, must be a CBS(0A), to signal the microcode to invoke the line request. Nothing is sent to the CRT until the CBS(0A) is issued.

Example,

assuming setup and prefill are complete.

CBS(0A)

This command is also used after usccessful RECALL or DEFFN' text entry to signal the controller to resume processing the line request.

NOTE: The 2200 will sometimes skip this command when it knows it will read in the data from the controller on the next command. This is a violation of protocol but the controllers live with it. Should the 2200 skip this command and go directly to CBS(0C) the controller should update the screen before allowing another Line Request to be started.

QUERY LINE REQUEST CBS(0B)

When an CBS(0B) command is received, the controller responds with one of the following IBS values.

00h -- No line request in progress.
01h -- line request still in progress.
0Dh -- line request terminated by CR.
FFh -- Recall key pressed (see note).
ENDI(XX) -- S.F. key pressed.

Note on recall:

After the FFh, the controller may send one or more bytes to the MVP. Each time the MVP sets CPB ready, the controller will send one more data byte with IBS. These are the characters from the entered text, read from right to left, beginning with the cursor position. The begnning of the buffer is indicated by ENDI. This sequence ends whenever the MVP stops setting CPB eady, sends OBS or CBS, or switches address. The controller should not clear the buffer when the 2200 has read all the bytes contained therein. Unfortunatly, the 2200 takes some shortcuts for expediency and may reread the buffer later.

Following the query, the MVP may do one of the following:

1. Nothing (another query later).

2. Delete line request (usually for HALT, and SF keys without parameters).

3. Refill -- this is more data to be merged to the present line request, as though the operator typed it (used for recall and DEFFN' quotes). Then End Line Request.

4. Terminate Line Request -- used to implement DEFFN' HEX(0D).

5. Error Line Request -- this beeps an error and continues the line request.

6. Ask for data.

ACCEPT LINE REQUEST DATA CBS(0C)

When a CBS(0C) is received after a line request has been completed, the controller will send the data. It should only be issued after a query has shown that the line is complete.

The controller sends the data if any, then an ENDI as terminator. if the ENDI is zero, the line request is complete; if 01h, the controller needs more time to finish updating the screen.

REQUEST CRT BUFFER CBS(0D)

This command causes the controller to check the CRT buffer of the current terminal. If it is empty, the appropriate status bit is set (address 02h = ready) to signal the fact. If not, then the controller will set the bit when the buffer does go empty.

REQUEST PRINT BUFFER CBS(0E)

This is just like the previous, except it refers to the current therminal's PRINT buffer, not CRT buffer.

ERROR LINE REQUEST CBS(0F)

This command causes the line request to resume, just like END-LINE-REQUEST, except it beeps first. It should not be used in conjunction with PREFILL or REFILL. It is normally used for undefined function keys.

TERMINATE LINE REQUEST CBS(10)

This command is used (after optional PREFILL or REFILL) to cause all the same actions as the operator pressing EXEC. It is normally used for the basic statement DEFFN' HEX(0D).

DEVICE STATUS CBS(11) IBS(xxxxxx) ENDI(00)

This command gets the controller type. Controllers which do not support this command will cause a timeout error in the 2200. When the timeout error is gotton the 2200 will assume that the controller is an MXD. The MXE and Triple controller support this command. The string returned by this command has the following format:

Byte 1	01	Triple controller without clock
	02	W/Time of Day clock
	04	W/Battery Backup TOD clock
	80	Needs Downloading (This bit may be set regardless of other bits)
Bytes 2 and 3		Code Revision:
Byte 2		Revision integer portion
Byte 3		Revision fractional portion times .01
Bytes 4 + ENDI(00)		Controller name

Start Clock CBS(12)

This command causes the board to begin issueing one second ticks to the 2200 at address 02h.

Download Code CBS(13) OBS(xxxx....)

This command signals the start of a block of RAM microcode for the controller. The 2200 does not care what the format of the block is. Each block will start with a CBS(13) and will be sent to the controller unbroken. It is assumed that the controller understands the format of the blocks.

End Download Code CBS(14)

This command signals the end of the downloading process. When the controller receives this command it will begin executing the downloaded code. Currently, the MXE is the only downloaded board. It expects to begin executing at the locations pointed at by location 4000h.

$MXE Command CBS(20) OBS(Command String)

This command has not been implemeted by the 2200 and is therefor not fully tested.
$MXE commands can be passed to the MXE preceeded by a CBS(20). Almost all the commands supported at the terminal end could be executed this way. The entire command string would be passed to the MXE without interpretation by the 2200.

Return Result of $MXE Command CBS(21)

This command would be used to get the result of an MXE command issued with CBS(13).

Enable Connect CBS(22)

This command enables the connect alert system for the currently selected port. This command sets a flag which the controller tests when a terminal connects to a device. If this bit is set (or the disconnect flag bit is set) then the controller alerts the 2200 to the connection through a connect flag at address 02h.
A connect is defined as the Data Set Ready signal on the RS232 line becoming active.

Disable Connect CBS(23)

This command disables the connect system for the currently selected port.

Enable Disconnect CBS(24) OBS(xxxx)

This command enables the disconnect alert system for the currently selected port. This command sets a flag which the controller tests when a terminal disconnects from a device. If this bit is set then the controller alerts the 2200 to the disconnection through a disconnect flag at address 02h.
The bytes xxxx define a timer value which is interpreted as follows:

> If xxxx = 0000 then the device is disconnected immediatly.
> If xxxx = FFFFh then any current disconnect countdown is canceled and the device is NOT disconnected by the controller.
> Otherwise, the value of xxxx is used a timer value and causes the controller to forcably disconnect the device after xxxx seconds.

NOTE: the controller disconnects the device by resetting DTR to it's inactive state until the 2200 allws reconnection by use of the command CBS(2C).

A disconnect is defined as the Data Set Ready signal on the RS232 line becoming inactive.
When a disconnect occurs AND the disconnect flag is set the controller should reset Data Terminal Ready to its inactive state for a period of 3 to 5 seconds so that any communication equipement connected to the port will let go of its communication line (such as a telephone line).

Disable Disconnect CBS(25)

This command disables the disconnect system for the currently selected port.

Alert 2200 to All Connected Ports CBS(26)

This command causes the controller to set the connect bit at address 02h for each port which is currently connected to an active device. This command does NOT set or reset any flags; it's action is one time only.

Clear Pending Disconnection CBS(27)

This command cancels a pending disconnection for the currently selected port but does not disable the disconnection detection system for that port.

Get ID of Terminal CBS(28) IBS(xxxx)

This command causes the controller to pass a 2 byte value to the 2200 which designates weather or not the currently connected device (at the currently selected port) supports RSD. The 2 bytes should be decoded as follows:

Byte 1: 00 = No RSD support
01 = Supports RSD

Byte 2: 00 = unknown terminal type
01 = 2336 DW terminal byte

Initiate Remote Screen Dump CBS(29) OBS(RSD parameters) ENDI(xx)

This command is used to begin a Remote Screen Dump from the currently selected port. If the device supports RSD then the controller returns a value of ENDI(01) to the 2200, initiates the RSD by sending the proper RSD start command to the device and passes the device the parameter string received from the 2200. If the device does not support RSD than the controller returns ENDI(00) to ther 2200 and discards the parameter string.

Parameters: N bytes to be sent to the terminal at the start of the screen dump which specify the area of the screen to be dumped.

xx: 00 = RSD cannot be done to this terminal.
01 = RSD begins

Abort Remote Screen Dump CBS(2A)

The 2200 issues this command to the controller when it wishes to stop a RSD already in progress. The Remote Screen Dump at the currently selected port is aborted and all RSD buffers are flushed. If no RSD is currently executing at the currently selected port the controller ignores this command.

Initiate Remote Screen Dump from Terminal Request CBS(2B)

This command is the same as CBS(29) except it is issued in responce to a RSD request issued by the device. The reason this is a seperate command is so that RSD requests can be queued up and the controller can keep track of them.

Reconnect Port CBS(2C)

This command instructs the controller to begin a 5 second timer which will then reconnect the currently selected device. This command is issued after the 2200 has completed all necessary disconnect procedures for a disconnecting device.

Select Special Package CBS(2D) OBS(xx) ENDI(yy)

This command selects a special support package such as Asynchronous Communications to be run at the currently selected port. The OBS byte is currently decoded as follows:

00h	NULL
01h	Standard Wang Terminal
02h	Asynchronous Communications Package (TC)

Set Controller Password CBS(2E) OBS(string)

The first six bytes of 'string' are set as the controller password.

Address 03h - Asynchronous Communications

Address 03h is only used by the MXE. It is used for communication between a Basic program tand the MXE Asynchronous Communications code. The user may use the $GIO command to perform input and output at this address useing the A## addressing convention. For a description of the protocol see the 'Asynchronous Communications Guide for 2236 MXE' Wang # 700-8089. Commands at this address may be broken at any time due to 2200 breakpointing. The user is not trusted as the 2200 is at other addresses and extensive error checking is done at this address. (at all other addresses the 2200 is TRUSTED not to send anything which will confuse the controller).

Address 07h - Send data to CRT

Address 07h is used to transmit characters onto the CRT of the current terminal. It should be ready whenever there is at least three bytes of space available. When it goes busy the CPU will time out (one millisecond) and service other partitions until a buffer-empty interrupt occurs. In addition to OBS with data to be displayed, this address also supports CBS's at this address. When the controller receives a CBS(xx) at address 07h it should place a FBh before it in the buffer. This will cause the terminal to accept this string as a command. The following are the three possible commands to the terminal:

1. Repeat

   FBh count character Where count is in the range 00h to 50h

   A CBS(04) to CBS(50), followed by an OBS of any byte except FB will have the effect of generating 4 to 80 characters of that byte. The code will be stored and transmitted in compressed form and thus will save time. It can be used to easily generate position cursor or tab functions. (CBS(00) to CBS(03) will work, but with a loss in efficiency).

   Example,

   OBS(01) CBS(05) OBS(0A)

   will place the string 01h FBh 05h 0Ah in the CRT buffer and will move the cursor to the beginning of row 5.

   *Note: The OBS must be sent without timeout.

2. Pause

   FBh delay where delay is in the range C0h to C9h

   A CBS(C0) to CBS(C9) will cause the terminal to pause for 0 to 1.5 seconds (one-sixth second increment). The controller will still continue to accept data until its buffer is full.

3. Character FBh

   The character FBh must be followed by the byte D0h so that the terminal will not interpret it as the start of a command. The controller places a D0h in the buffer immediatly after storing the FBh.

Address 04h - Send Line to Slave Printer

Address 04h is similar to address 07h except that the characters are directed to the printer of the current terminal. This may be used in VP mode, as well as MVP mode, but then only with terminal #1. The terminal does not support pause or repeat to the printer but all FBh must be followed by D0h as at address 07h.
MXE only: Remote Screen Dump bytes are transfered to the 2200 at this address. Whenever the 2200 sets CPB ready at address 04h the MXE will send RSD bytes to the 2200 followed by an ENDI(00) when the buffer is empty.

Address 02h - Receive Controller Status

Address 02h is used to report the stati of the various terminals to the 2200. When enabled by address 02h with CPB ready the controller will send from 7 to 11 bytes of data and 1 ENDI data byte to be used as a terminator for the input sequence.

Bytes	Explanation	
1	Low nibble:	RESET flags (1-bit per terminal)
	High nibble (MXE only):	Connect flags
2	Low nibble:	Halt/step flags (1-bit per terminal)
2	High nibble (MXE only):	Disconnect flags

Note: The above 2 bytes are organized so that the 01 & 10-bits represent terminal 1, 02 & 20-bits terminal 2, 04 & 40-bits represents terminal 3 and the 08 & 80-bits terminal 4.

3 One second clock tick as follows (MXE and Triple controller only):

If the 2200 has commanded this controller to tick of seconds then it will interrupt the 2200 at each one second tick and will pass the number of seconds which have elapsed since the 2200 last read the board status in this byte. The 80-bit must be set for the 2200 to consider this count legal. It is legal for the 2200 to not read the status for as long as it wants and the board must maintain the elapsed time in this byte. A board should only use set the 80-bit in this byte if the 2200 has previously issued command CBS(12).

4a	Terminal 1 status
4b (MXE)	Terminal 1 request
5a	Terminal 2 status
5b (MXE)	Terminal 2 request
6a	Terminal 3 status
6b (MXE)	Terminal 3 request
7a	Terminal 4 status
7b (MXE)	Terminal 4 request

Note: The above 8 bytes are organized as follows:

Byte a:
Bit
- 01 - Print buffer became empty (requested)
- 02 - CRT buffer became empty (requested)
- 04 - Keyboard buffer not empty or Line Request complete
- 08 - RSD buffer almost full
- 10 - unused
- 20 - unused
- 40 - unused
- 80 - Request byte (byte b) follows this byte

Byte b:
This byte is a one byte request to the 2200 for some function to be performed. It is only sent to the 2200 when the terminal has a request to make. The following requests are currently defined:

00h	Null
01h	Request Remote Screen Dump to be performed
02h	Cancel currently executing RSD

2200 Input/Output Bus

This section describes the signals on the 2200 I/O bus and their uses. Those signals which do not pertain to the functioning of the 2200-controller protocol have been omitted (such as ground). The list of bus signals is as follows:

RESET	Reset strobe - used by the controller in VP mode to reset the CPU
PRIME	Halt/Step strobe - used by the controller in VP mode to Halt or Step the CPU
Address bus	8 bit address bus controlled by the 2200
ABS	Address Bus Strobe
Input Bus	8 bit data bus from the controller to the 2200
ENDI	ENDI bit - 9th data bit in the direction of from the controller to the 2200 only
IBS	Input Bus Strobe - from the controller to the 2200
Output Bus	8 bit data bus from the 2200 to the controller
OBS	Output Bus Strobe - from the 2200 to the controller for data bytes
CBS	Control Bus Strobe - from the 2200 to the controller for control bytes
CPB	CPU Busy - Signal (not a strobe) from the 2200 signaling to the controllers that the 2200 is ready to input bytes.
Ready/Busy	Ready-Busy signal to the 2200 from the controller

The 2200 selects a controller by setting the Address Bus to the controller address and strobing ABS. All controllers not selected should then not use the bus in any way so that the 2200 can communicate with the selected controller. The Ready/Busy line is set by the controller whenever the controller is selected. If more than one address is found on the same controller the R/B signal is set to the selected sub-address.

The Output Bus is used by the 2200 to output bytes to the controller using the OBS and CBS strobes. The controller can distinguish the two strobes apart thus allowing both data bytes and control bytes to be sent to the controller.

The Input Bus is used to transfer data bytes from the controller to the 2200 with the IBS strobe. A ninth data bit is used in this direction called the ENDI bit. This bit is used to indicate special conditions such as end of data transfer, etc. The ENDI bit is actually a flag in a register in the 2200 rather than a strobed data bit.

CPB is set by the 2200 to indicate its readiness to accept bytes from the controller. With a few exceptions the controller must respond within 15MS or a timeout error will occur. Once the CPB signal is set either bytes are transfered or the timeout condition is encountered.

The RESET strobe is used by the terminal controller in VP mode to reset the CPU. This signal should not be strobed in MVP mode or the system will crash.

The PRIME signal, also called HALT/STEP is strobed by the terminal controller to signal the deesire to Halt between instructions or step through the next signal. This signal is used in MVP mode only.

MXE/SVP Option W RS-232c Connection Pinout

Pin #	EIA	CCITT	Name	Signal Source
1	AA	101	Protective Ground	
2	BA	103	Transmit Data	DTE
3	BB	104	Receive Data	DCE
4	CA	105	Request to Send	DTE
5	CB	106	Clear to Send	DCE
6	CC	107	Data Set Ready	DCE
7	AB	102	Signal Ground	
8	CF	109	Received Line Signal Detector*	DCE
9			Unconnected	
10			Unconnected	
11			Unconnected	
12			Unconnected	
13			Unconnected	
14			Unconnected	
15			Unconnected	
16			Unconnected	
17			Unconnected	
18			Remote Analog Loopback*	DTE
19			Unconnected	
20	CD	108.2	Data Terminal Ready*	DTE
21			Remote Digital Loopback*	DTE
22	CE	125	Ring Indicator*	DCE
23	CH/CI	111/112	Data Signal Rate Selector*	DCE/DTE
24			Unconnected	
25			Unconnected	

* This pin is not connected on port 1 of an SVP Option W (Ports 2 and 3 utilize these signals)

Baud Rates:

The MXD and Triple controller baudrate switches are usually marked to indicate the desired settings. For older controllers it may be necessary to consult the 2236 terminal guide for setting.

Below is a list of baud rates available on the MXE and SVP controllers. Some of the rates can only be set via the $MXE baud rate command and will not have an entry in the second column. Others can be set by the hardware baud rate switches. Please see the discussion following the table which describes how default hardware baudrates are set. When setting baudrates via the $MXE command, the baud rate should be entered exactly as found in this table.

Rate	Hardware switch setting
50	
75	
100	
110	0
134.5	1
150	2
200	3
300	4
600	5
1200	6
undefined	7
2400	8
undefined	9
4800	A
undefined	B
9600	C
19200	D
undefined	E
see note	F

NOTE: Setting all the hardware switchs to X'FF' will cause the MXE to go into diagnostic burn in mode. In this state the MXE will continuously loop on its board diagnostics and will not talk to the 2200 or allow any user input. All board errors found will be displayed on the screen.

DOCUMENT SUMMARY

Document Id: 0019E
Document Name: MXE commands
Operator: Eric
Author: Eric

Comments:

Pages to be printed 4

Notify U13 on system PAM.

To: Bruce Patterson
From: Eric Wilson
Subject: MXE commands
Last revised: 07/27/82
Pages: 4 (including cover)

- -

The following is a BRIEF description of the MXE command structure. By no means is this the final version. I submitting this only for the purpose of obtaining feedback.

Your comments are welcome and should be directed to Eric Wilson at X2192, M/S 1383 in Lowell.

c.c. Neeraj Sen

The following is a list of MXE commands from the 2200 to the MXE. Also listed are those commands which are currently supported by the MXE only:

HEX CODE	DESCRIPTION
00	Null
01	Poweron Sequence
02	Initialize Current Terminal
03	Delete Line Request
04	Keyboard Ready Check
05	Keyin Poll Request
06	Keyin Line Request
07	Line Request
08	Prefill Line Request
09	Refill Line Request
0A	End of Line Request
0B	Query Line Request
0C	Accept Line Request Data
0D	Request Crt Buffer
0E	Request Print Buffer
0F	Error Line Request
10	Terminate Line Request
11*	Device status
12*	Start clock
13*	Download code
14*	End download
15	
16	
17	
18	
19	

Commands pertaining to MXE only are greater than 1F:

20	$MXE command
21	Return result of $MXE Command
22	Enable Connect
23	Disable Connect
24	Enable Disconnect
25	Disable Disconnect
26	Alert 2200 to ALL connected ports
27	Clear pending disconnection
28	Get ID of terminal
29	Initiate Remote Screen Dump
2A	Abort Remote Screen Dump
2B	Initiate Remote Screen Dump in responce to terminal request
FF	Select Terminal

* Command not implemented in MXD.
** Not yet implemented.

Command Descriptions

28 - Get ID of terminal:

CBS(28) IBS(2 bytes)

Byte 1: 00 = No RSD support
01 = Supports RSD

Byte 2: 00 = unknown terminal type
01 = 2336 DW terminal byte

29 - Initiate Remote Screen Dump:

CBS(29) OBS(RSD Parameters) ENDI(byte)

Parameters: N bytes to be sent to the terminal at the start of the screen dump which specify the area of the screen to be dumped.

Byte: 00 = RSD cannot be done to this terminal.
01 = RSD begins

2A - Abort Remote Screen Dump:

CBS(2A)

Remote Screen Dump is immediatly ended and any bytes currently in the buffer of the MXE are flushed

Remote Screen Dump:

RSD bytes are read by the 2200 at address /03. Each time the 2200 wants RSD bytes from the MXE it sets CPB ready at address /03 of the MXE after checking for ready. The MXE sets bit 08 in the termstat byte of that port whenever it has bytes ready for the 2200 (as is done for the CRT at address 07). The MXE signals the end of RSD by outputting ENDI(FF) after all the RSD bytes have been taken by the 2200. Each time the 2200 empties the RSD buffer in the MXE the it will send ENDI(00) to signify an empty buffer but NOT THE END OF RSD!

DOCUMENT SUMMARY

Document Id: 0021E
Document Name: MXE Description
Operator: Eric
Author: Eric

Comments: MXE Specification

Pages to be printed 14

Notify U13 on system PAM.

2236 MXE SOFTWARE SPECIFICATION
(Second draft)

Eric Wilson
Jan,22,1982

MXE DESCRIPTION

The 2236MXE is an intelligent multi-terminal controller which is downward compatible with the 2236MXD terminal controller. In addition to emulating all MXD functions exactly the MXE will have the following additional characteristics:

- 48K ram will be loaded with applications programs (such as editors) thus allowing changes to be easily made to MXE operation. As prom will have only poweron diagnostics, bootstrap, and basic I/O, it should not need to be changed to support foriegn terminals and new devices.

- Various editors and applications programs can be loaded into ram at any time to suit specific applications

- The standard editor (which will emulate the MXD editor) will change add an insert mode to the current space insert mode. This new insert mode will allow characters to be directly inserted in the line without having to first insert blanks.

- An MXE command mode will be implemented to allow the user to alter parameters within the MXE and run MXE diagnostics. MXE command mode will be entered by pressing the "LOAD" key three times. The MXE will then ask for a command which will not be passed on to the 2200. This mode will be extremly helpful to customer engineering for diagnostic purposes.

- MXE command mode will be protected by a password system. The 2200 will pass a password to the MXE during system configuration. This password must then be wntered to use most MXE commands. The default password is "SYSTEM" and will be used until the 2200 passes the MXE a password.

All MXE commands follow a general format as follows:

'Command Name' 'password' 'parameter1' 'parameter2' 'parameter3' 'return'

Where:

'Command Name' = one character command abbreviation

'password' = six character password following these rules:
1) No imbeded backspaces or returns, and
2) The first character must not be a space.

'parameter#' = command parameter as specified in each command description

'return' = return key on keyboard

Blanks may be used anywhere in the command with the following exceptions:

1) Parameters which include test strings may not have extraneous imbeded blanks, and

2) the command name and the password must be separated by only one blank.

The following is a list of MXE commands and their format:

code	function	parameters
P	Jump to poweron Sequence.	NOTE: this command will reinitialize the MXE. The current state of the board will be completely lost.

S Set mode. Additional parameters are required to specify what is to be set:

- Txx — Select port xx as subject of next set command.
- Bxx — Select Baud Rate. xx is index into baud rate table, not actual rate.
- Ppsw — Set password to psw which consists of any 8 keyboard enterable printable characters.
  NOTE: Password is intially set to six blanks and it is suggested that it be set by the superuser or the MVP when loading the MXE editor.

Xxx Exchange Superuser (initially port one) with port xx

Lxxxxyyyy zz Load xxxx bytes starting at location yyyy in ram. Code will be loaded from port zz or terminal issuing command if zz is not specified.

Jxxxx Jump to location xxxx. NOTE: If the routine being jumped to will later return to MXE command mode it must save the return location which is passed in index register IX.

E Display error history. A short history of parity errors is kept in a FIFO stack. As errors occur they are flagged on port one and stacked. Errors are pushed off the end of the stack when it is full.

D Dump status of MXE board. This command may not be implemented depending on space considerations. The status of various board elements (darts, registers, etc) is displayed.

** NOTE: x, y, and z are always in hexadecimal form.
*** NOTE: psw is a password consisting of 8 keyboard enterable printable characters

MXE commands originating from the MVP have the following format (please see complete description of each command following this table):

HEX CODE	DESCRIPTION
00	Null
01	Poweron Sequence
02	Initialize Current Terminal
03	Delete Line Request
04	Keyboard Ready Check
05	Keyin Poll Request
06	Keyin Line Request
07	Line Request
08	Prefill Line Request
09	Refill Line Request
0A	End of Line Request
0B	Query Line Request
0C	Accept Line Request Data
0D	Request Crt Buffer
0E	Request Print Buffer
0F	Error Line Request
10	Terminate Line Request
11*	Select Baud Rate
12*	Exchange Superuser
13*	Set Password
14*	Download Code
15*	Jump to Location in Memory
16*	Dump Status
FF	Select Terminal

* Command not implemented on MXD.

the above commands to the MXE are received at MXE board address 06.

HEX(01) - Poweron Sequence

Restart the MXE as at Poweron; Diagnostics are rerun, buffers and devices are reinitialized, crt's are cleared, and the mode becomes VP mode.

HEX(02) - Initialize Current Terminal

Reinitialize the currently selected terminal. Buffers are cleared, flags and pointers are initialized, and the crt is cleared.

HEX(03) - Delete Current Line Request

The pending line request to the current terminal is cleared along with the terminals input buffer.

HEX(04) - Keyboard Ready Check

If the terminal input buffer of the currently selected terminal is empty, the MXE will send HEX(00) on IB1-IB8, if there is a character in the buffer it will send a non-zero value.

HEX(05) - Keyin Poll Request

If there is a character in the current terminal's input buffer it is sent (with ENDI if it's an SF key), otherwise HEX(00) is sent.

HEX(06) - Keyin Line Request

If there is a character in the current terminal's input buffer it is sent, otherwise HEX(00) is sent and a Keyin Line Request is set up causing the next input character to generate a completed line request.

HEX(07xxxxyyzz) - Line Request

Set up line request for a field of up to xxxx characters (hex representation of the count, not to exceed 480 characters) starting from the current cursor position on the current terminal. Field limits are strictly enforced. A Line Request remains active until a carriage return or a special function key is entered, or until a Deleted Line Request command is issued (as in reset or halt). yy is defined as follows; The 80-bit specifies underline, the 04-bit specifies Edit mode, and the 01-bit specifies that the terminal input buffer should be flushed (thereby ignoring any keystrokes entered previous to the Line Request). Characters flushed from the buffer are not echoed on the crt. zz specifies cursor position (columnwise).

HEX(08xxxx...) - Prefill Line Request

This command is optional and must follow a Line Request command (HEX(07)). The current line request is prefilled, starting on the left, with the text following the HEX(08) code and proceeding until a CBS strobe which accompanies an End of Line Request (HEX(0A)) command. The cursor is left at the leftmost position of the request and the prefill string is treated as keystrokes.

HEX(09xxxx...) - Refill Line Request

Identical to a Prefill Line Request (HEX(08)) except that the cursor is not repositioned at the begining of the prefill string. The prefill is treated as keystrokes.

HEX(0A) - End of Line Request Sequence

This command signals the end of a Prefill Line Request. The prefill string is not displayed on the crt until this command is received.
This command is also used after successful RECALL or DEFFN' text entry to signal the MXE to resume processing the Line Request.

HEX(0B) - Query Line Request

The MXE responds to this command with one of the following:

00 - No Line Request in progress
01 - Line Request still in progress
0D - Line Request terminated by carriage return
FF - Recall key pressed *
ENDI+xx - SF key pressed

* Recall note: After HEX(FF) the MXE can send one or more bytes to the MVP. Each time the MVP sets CPB ready, the MXE will send one data byte. The bytes sent are from the line request buffer in reverse order (starting at the cursor position). The begining of the buffer is denoted by an ENDI. This continues until the MVP stops setting CPB ready, sends OBS or CBS, or switches address.

Following the Query the MVP may do one of the following:

- Nothing (query again later).
- Delete Line Request (for Halt, SF key, etc).
- Refill - this string is added to the terminal input buffer and treated as keystrokes. An End Line Request follows.
- Terminate Line Request - used to implement DEFFN' HEX(0D).
- Error Line Request - beeps an error and continues Line Request.
- Ask for data.

HEX(0C) - Accept Line Request

After a Line Request is completed the MXE will send the line. This command should only be used after a Query has found the Line Request finished. An ENDI is sent as terminator. If the ENDI is HEX(00) the Line Request is finished, if it is HEX(01) the MXE needs more time to finish updating the crt.

HEX(0D) - Crt Buffer Request

If the crt buffer is empty the MXE sets address HEX(02) ready, otherwise it sets address HEX(02) ready if/when the buffer goes empty.

HEX(0E) - Print Buffer Request

Identical to the Crt Buffer Request except that it refers to the print buffer.

HEX(0F) - Line Request Error

This command causes the current line request to resume after beeping. It should only be used for Line Requests without pre- or refill. It is usually used for undefined function keys.

HEX(10) - Terminate Line Request

This command causes all the same actions as when the operator presses Exec. It is normally used for DEFFN' HEX(0D).

HEX(11pswxx) - Select Baud Rate**

This command selects a baud rate for the currently selected terminal. xx is an index into the Baud Rate Table in prom. psw is the current password and must be sent. The Baud Rate Table is as follows:

Index(HEX)	Equivalent Baud Rate (bps)
00	110
01	300
02	600
03	1200
04	2400
05	4800
06	9600
07	19200

HEX(12pswxxyy) - Exchange Superuser**

Port xx is swapped with port yy.

NOTE: The effect of the swap is to give port xx the partition of port yy. The effect is the same as physically swapping the board connections!!

HEX(13pswuuuuuu) - Set New Password**

This commands changes the old password psw to a new password uuuuuu.

HEX(14pswxxxxyyyyzz) - Load Code**

xxxx bytes of are to be loaded starting at location yyyy. If zz is specified as FF the bytes are loaded from the 2200 bus, otherwise they are loaded from port zz (with 00 being port one, and so on until port four).

HEX(15pswxxxx) - Jump to Location in Memory**

A jump to location xxxx is executed. If the code at location xxxx will later execute a return it must save the return address passed in register IX.

HEX(16xx) - Return Information**

The information requested by code xx is returned to the MVP. The request codes are as follows:

CODE (HEX)	INFORMATION RETURNED
01	Baud rates of all the ports.
02	Everything known about each port
03	Revision numbers of ram and prom code
04	Error history

The return for this command is of the form:

yyxxx...

Where: yy is the number of bytes being returned.
xxx.. is the stream of bytes.

HEX(FFxx00) - Select Terminal

Terminal xx is selected as the current terminal to which all subsequent commands will apply.

Possible selections are: 00 to 03 for ports,
and FF for MXE.

NOTE: In the case of commands directed to the MXE which require a large response, the MXE will be treated (logically) as another port with its own buffers. So, to receive the information requested by an MXE command the MXE must be selected in the same manner as MXE ports are selected. (All commands already implemented on the MXD will be emulated exactly)

** This command not implemented on the MXD.

BOARD ADDRESSES

Each MXE Controller Board can be accessed through several different addresses. The high order nibble selects the board while the low order nibble selects an address on the selected board. Possible high order nibbles are as follows:

HEX(0)
HEX(4)
HEX(8)
HEX(C)

The seven addresses on each board serve specific functions as follows:

ADDRESS (hex)	FUNCTION
01	Used only in VP and bootstrap mode for the superuser's keyboard. This address is always busy in MVP mode.
02	Terminal controller status.
03	Unused
04	Printer of currently selected terminal (also used in VP and Bootstrap modes).
05	Used only in VP and bootstrap mode for the superuser's crt. NOTE: When this address is enabled in MVP mode, ths MXE will enter VP mode.
06	Control operations and output from controller. All MXE commands are issued to this address. NOTE: When this address is enabled in VP mode, the MXE will enter MVP mode.
07	Currently selected terminal's crt.

MODES OF OPERATION

MXE SET-UP

When the system is powered on the MXE first does extensive diagnostics on its various components. Parity is tested throughout memory, all Z-80 registers are checked out, the CTC and DART chips are verified as accurate, and the 2200 bus registers are checked for errors. If the board has passed all its, tests it then proceeds onto VP/BOOTSTRAP mode. If not, any malfunctions found will be displayed on the terminal at port one (or, if that port is not functioning, the next highest functioning port) and, depending on the malfunction, the MXE either stops running or goes into VP/BOOTSTRAP mode.

VP/BOOTSTRAP MODE

After the diagnostics routine, if a fatal malfunction has not occured, the MXE enters VP/Bootstrap mode. It initializes all its buffers and pointers, and sets up the CTC timers and DART chips. It then sets the 2200 bus ready and begins its basic polling loop. Only the terminal at port one can operate in this mode (if port one is not working the next highest functioning port is utilized). The MXE does not perform editing in this mode. Input is passed on to the MVP until an editor is loaded and the mode becomes MVP. All MXE commands can be entered in this mode. MXE commands are identified by an escape sequence which informs the MXE not to pass them on to the MVP. The escape sequence is as follows:

In this mode the RESET and HALT functions from the superuser (port one, etc.) actually cause strobes on the I/O bus (Prime and Halt lines). This never occurs in MVP mode.

MVP MODE

MVP mode is entered when communication from the MVP is received at addresses other than HEX(01) or HEX(05). In this mode all ports are fully operational. HOWEVER, apart from backspace, no editing functions will operate until an editor is loaded and running!

BUFFERS

Each port has associated with it four buffers as follows:

Buffer	Size (bytes)	Use
Keyboard	32 - 2byte entries	To store all keystrokes received from the keyboard. If no line request is pending these keystrokes will remain in the buffer until: - A line request or keyin command is issued. - The buffer is emptied by an MVP command. NOTE: Keystrokes are lost after the buffer is full!
Line Request	480	Bytes entered in response to a line request are stored in this buffer.
Printer	160	Bytes to be sent to the printer are stored in this buffer.
Crt	250	Bytes to be sent to the crt are stored in this buffer.

Index(HEX)	Equivalent Baud Rate (bps)
00	110
01	300
02	600
03	1200
04	2400
05	4800
06	9600
07	19200

DOCUMENT SUMMARY

Document Id: 0026E
Document Name: MXE Modes
Operator: Eric
Author: Eric

Comments: Mode & addr descr

Pages to be printed 4

Notify U13 on system PAM.

VP Mode (Prom)

VP (or Bootstrap) mode is entered whenever the system powers on or the 2200 enables the MXE at addresses 01 or 05 or poweron diagnostics are rerun.

In VP mode the 2200 communicates with terminal one (logical terminal one) only and does all editing itself. The MXE acts only as a buffer between the two (but does search the input stream for the MXE escape code for MXE commands from the terminal). Whenever the keyboard buffer holds a keystroke the MXE sets address 01 ready. Addresses 05 and 04 follow the states of the crt and printer buffers (respectively). When the buffer is empty the address is set ready. The MXE continues to set the address ready until the buffer is full. The address is then set busy until the buffer goes empty again.

Addresses 07, 03, and 02 remain busy in VP mode.

Address 06 is kept ready whenever the MXE is ready to process MXE commands from the 2200. Unlike the MXD, the MXE does not automatically enter MVP mode when enabled at an address other than 01, 04, or 05. A subset of MXE commands are allowed in VP mode (generally those which control functions other than editing or input from terminals, etc.). The MXE will set up to receive a command from the 2200 whenever 06 is enabled with CBS set. The MXE is expected to output to the 2200 when enabled with CPB set at address 06 (some commands require a response). CBS is used (generally) to denote the begining and end of a control sequence.

MVP Mode (Ram)

MVP mode is entered when the 2200 issues the command to transfer control to RAM after the MXE has been downloaded (a jump is performed to some location in RAM). In MVP mode the MXE functions as both editor and multiplexer for the 2200.

Address 06 is used for MXE commands and output from the terminals and MXE. The status of 06 (R/B) is determined by the status of the currently selected terminal.

Addresses 04 and 07 are used for the currently selected terminals printer and crt respectively.

Address 02 is the status address. It remains busy until one of the following occurs:

- any crt buffer goes empty,
- or any printer buffer goes empty,
- or a line request completes,
- or a Reset or Halt/Step is keyed,
- or the Time of Day clock on the MXE board reaches a second.

When the 2200 enables address 02 (with CPB set) the MXE returns several status bytes which denote the status of port buffers, TOD clock, etc.

Address 03 is not used in MVP mode and therefor remains busy all the time.

Addresses 01 and 05 follow the states of the current superusers keyboard and crt. When the MXE is enabled at either of these addresses the MXE will perform a jump to a location in PROM thereby returning to VP mode.

Address uses in VP mode*

Address (logical/bit)	Use	Status
01/01	Terminal one keyboard	Ready when terminal ones keyboard holds a keystroke.
02/02	Board status	Busy in VP mode.
03/04	Unused	Always busy.
04/08	Terminal one printer	Ready when terminal ones printer buffer is empty. Remains ready until buffer goes full. Goes ready again when buffer goes empty.
05/10	Terminal one crt	Ready when terminal ones crt buffer is empty. Remains ready until buffer goes full. Goes ready again when buffer goes empty.
06/20	Command address	Ready when the MXE is ready to receive a command.
07/40	Unused	Unused in VP mode
08/80	Unused	No hardware support for this address.

* NOTE: The above definition are different than those for the MXD in VP mode!

Address uses in MVP mode

Address (logical/bit)	Use	Status
01/01	Terminal one keyboard	Ready when terminal ones keyboard holds a keystroke.
02/02	Board status	Ready when: - any crt buffer goes empty, - or any printer buffer goes empty, - or a line request completes, - or a Reset or Halt/Step is keyed, - or the Time of Day clock on the MXE board reaches a second.
03/04	Unused	Always busy.
04/08	Current printer	Ready when the currently selected terminals printer buffer is empty. Remains ready until buffer goes full. Goes ready again when buffer goes empty.
05/10	Terminal one crt	Ready when terminal ones crt buffer is empty. Remains ready until buffer goes full. Goes ready again when buffer goes empty.
06/20	Command address	Ready when the MXE is ready to receive a command.
07/40	Current crt	Ready when the currently selected terminals crt buffer is empty. Remains ready until buffer goes full. Goes ready again when buffer goes empty.
08/80	Unused	No hardware support for this address.

DOCUMENT SUMMARY

Document Id: 0030E
Document Name: $MXE command
Operator: Eric
Author: Eric

Comments: Commands + baud rate

Pages to be printed 14

Notify U13 on system PAM.

To:
From: Eric Wilson
Subject: MXE command mode
Date: 03/19/82
Pages: 5 (including cover)

- -

This document is a preliminary, IN HOUSE release describing the $MXE command mode as it is currently implemented in the first MXE prom release.

Your comments are welcome and should be directed to Eric Wilson at X2192, M/S 1383 in Lowell.

c.c.

$MXE COMMAND MODE:

To enter $MXE command mode key three "LOAD" keys in succession (typing LOAD will not work. The MXE must receive three LOAD ATOMS). The prompt message "ENTER MXE COMMAND:" followed by a new line of "%" will be printed on the terminal. The user should then enter the desired $MXE command (as described on the following pages) followed by a return. The MXE will process the command and prompt for another command until a blank line is entered thus putting the user back in the previous mode. The next MXE command should not be entered until the MXE prompts for it. Keying extra 'RETURN's to try to speed up the processing of MXE commands will only serve to slow down the command processing and take the user out of MXE command mode after the current command!

Any terminal may enter $MXE COMMAND MODE at ANY time. If the 2200 is printing to the screen during MXE command mode the 2200 output will be temporarily suspended to prevent the two outputs becoming intermixed!

I/O BUS SPECIFICATION:

$MXE commands may be issued from the 2200 at address 06 as follows:

CBS(20) OBS('command string') CBS(00)

Where 'command string' is of the same format as $MXE commands issued from the terminal.

The MXE will return the result of the command at address 06 followed by a 00 byte with ENDI if the 2200 sends a CBS(21) and sets CPB ready. The 2200 must take the entire result at once. If the 2200 breaks while taking the result, the rest of the result will be lost.*

* NOTE: in the bootstrap all return codes are one byte long followed by a byte with ENDI on. A return code of '00' denotes no errors.

MXE COMMANDS:

In the command descriptions that follow, please use these definitions:

'psw' is a six character password containing no blanks (the default is 'MXEPSW'), and
'port designator' is a one character designator as follows:
0 = the port at which the command is being typed (this is so that the user need not know which port s/he is connected to), and
1, 2, 3, 4 are the absolute MXE physical port addresses.

All commands begin with a ONE BYTE command code. Most commands then have a sixbyte password which is the MXE password (similar in use to the 2200 system password) followed by any needed parameters in the order specified below. The user can always type help (while in MXE command mode) to obtain a list of command codes.
The command line is devided into fields. The first field is one character long and is the 'COMMAND BYTE' field; it should be used for the one byte command. The second field is the password field. it is six characters long and begins at the third column (thus allowing one blank after the command byte). When the password is typed in the password field it will not be printed on the screen. Instead, the MXE will print the numbers 0 through 5 as characters are typed. this is so that the password can be protected. After the password field is a free format field in which the user types the rest of the command. The MXE is blank insensitive so the command parameters may be typed as the user wishes with the following exceptions:

1) A blank must not be found in a baud rate specification, and
2) The string that is to be printed on all screens (command 'G') must be typed exactly as it is to be printed.

Commands:

A) Set primary user.
Format: A 'psw' 'port designator'

The port desinated becomes the new primary user in VP mode

B) Set baud rate.
Format: B 'psw' 'port designator' 'baudrate'

The baud rate specified is set at the port designated

C) Set password.
Format: C 'psw' 'newpsw'

'newpsw' becomes the new password

D) Download code.
Format: D 'psw' 'port designator'

Z-80 microcode is loaded into the MXE at the port designated. The protocall is as follows:

If the command is not legally specified the MXE will send an 'ILLEGAL COMMAND' to the port originating the command (which may also be the port designated). Otherwise, the MXE will ask the user for a confirmation of the command. If the user responds affirmativly, the MXE will output a X'00' at the designated port when it is ready to start receiving code. It will then expect to receive ten bytes of X'00' in succession followed by code in the following format:

'2-byte destination' '1-byte count (n)' 'n bytes of code'

The MXE will continue taking in strings of this form until a count of zero is received at which time the MXE will jump to the address contained in location X'4000'. No form of error detection exists in this format once the downloading has begun so the user should be very careful to not get out of synch with the MXE. Also, all other MXE activity will halt while this command is in progress. It should be noted that the string of 10 zero bytes is needed for protection against a user designating an incorrect port address or the accidental invocation of this command.

E) Analog loopback
Format: E 'psw' 'port designator'

Analog loopback is executed at the port designated. This command will cause all I/O at all ports to be temporarily suspended!

F) Digital loopback
Format: F 'psw' 'port designator'

Digital loopback is executed at the port designated. This command will cause all I/O at all ports to be temporarily suspended!

G) Print to all screens
Format: G 'message'

Print 'message' on all terminals connected to this MXE. 'message' must not run over the 78 byte limit of the command. 'message' will be inserted in the output stream of all terminals regardless of their state and a 3 second pause will be executed to allow all users to read the message.

H) Help
Format: H

A list of all $MXE commands is printed on the terminal

I) Memory test
Format: I 'psw'

All of RAM is tested in a nondestructive way. Although all port activity on the MXE will slow down somewhat, no damage will be done.

J) Restart
Format: J

This command restarts the MXE at its poweron diagnostics. This command can only be issued while in the bootstrap. The current state of the MXE will be completely reinitialized! All baud rates which have been set through $MXE command mode will be reset to their hardware defaults!!.

L) Lock
Format: L

This command locks the current baudrate of the port issueing it. No port may change the baud rate of a port which is locked. This command is a toggle. Each time it is issued the state of the baud rate lock will be reversed.

NOTE: During many of the above commands the performance of the MXE will be impaired. Generally, this consists of all the ports slowing down somewhat. But in the case of analog and didgital loopback ALL port I/O will cease until the loopback is completed. Therefor, it is absolutly imperative that all users connected to the MXE when loopback is to be performed be notified that the MXE will not be functioning smoothly.

Baud Rates:

Below is a list of baud rates available on the MXE. Some of the rates can only be set via the $MXE baud rate command and will not have an entry in the second column. Others can be set by the hardware baud rate switches. Pleas see the discussion following the table which describes how default hardware baudrates are set. When setting baudrates via the $MXE command, the baud rate should be entered exactly as found in this table.

Rate	Hardware switch setting
50	
75	
100	
110	0
134.5	1
150	2
200	3
300	4
600	5
1200	6
undefined	7
2400	8
undefined	9
4800	A
undefined	B
9600	C
19200	D
undefined	E
see note	F

NOTE: Setting all the hardware switchs to X'FF' will cause the MXE to go into diagnostic burn in mode. In this state the MXE will continuously loop on its board diagnostics and will not talk to the 2200 or allow any user input. All board errors found will be displayed on the screen.

How to set hardware baudrate defaults:

Two eight bit baudrate selection switches exist on the MXE. Each bank of eight bits is devided into two four bit groups each of which cooresponds to a specific MXE port. To find which group is used for each port hold the MXE board with the rail in your right hand and the two banks of baudrate switches facing you at the bottom left corner of the board. There will actually be three banks of switches in this corner; the two eight bit banks mentioned above and a smaller four bit bank which is used for setting the boards I/O bus address. Disregard this smaller bank for the purpose of this discussion.

The four ports on the MXE are numbered 1 to 4 with the port closest to you being 1 and the farthest one being 4. The switch banks are devided with the top right four bits cooresponding to port 1, the top left four bits cooresponding to port 2, the bottom right to port 3 and the bottom left to port four.

Once you have located the group of switches for the desired port, find the switch setting in the table above and set the switch with the most significant bit to the left and least to the right. Notice that the bits will be ON when the switch is positioned toward you (as marked on the switch). The baud rate chosen for a port must match the baud rate of the device connected to the port. If this is not the case, communication will not be possible between the device and the MXE.

Parity errors:

The MXE hardware is equiped with parity checking in its Random Access Memory (RAM). Each time a byte is read from RAM the parity on that byte is tested. If the parity bit is incorrect the MXE will perform a parity interrupt which will cause it to return to the bootstrap no matter what it is doing at the time. If it is already in the bootstrap it will restart the bootstrap. All data will be lost in this situation.

At this point the user may enter $MXE command mode and run specific diagnostics or may restart the MXE at its diagnostic bootstrap. The diagnostic bootstrap will diagnose the board completely (as it does each time the board is powered up) and list whatever error it finds on all screens. It is NOT possible to return to the code in RAM without downloading new code into the MXE first!

To:
From: Eric Wilson
Subject: MXE command mode
Date: 03/19/82
Pages: 5 (including cover)

- -

This document is a preliminary, IN HOUSE release describing the $MXE command mode as it is currently implemented in the first MXE prom release.

Your comments are welcome and should be directed to Eric Wilson at X2192, M/S 1383 in Lowell.

c.c.

$MXE COMMAND MODE:

To enter $MXE command mode key three "LOAD" keys in succession (typing LOAD will not work. The MXE must receive three LOAD ATOMS). The prompt message "ENTER MXE COMMAND:" followed by a new line of "%" will be printed on the terminal. The user should then enter the desired $MXE command (as described on the following pages) followed by a return. The MXE will process the command and prompt for another command until a blank line is entered thus putting the user back in the previous mode. The next MXE command should not be entered until the MXE prompts for it. Keying extra 'RETURN's to try to speed up the processing of MXE commands will only serve to slow down the command processing and take the user out of MXE command mode after the current command!

Any terminal may enter $MXE COMMAND MODE at ANY time. If the 2200 is printing to the screen during MXE command mode the 2200 output will be temporarily suspended to prevent the two outputs becoming intermixed!

I/O BUS SPECIFICATION:

$MXE commands may be issued from the 2200 at address 06 as follows:

```
CBS(20) OBS('command string') CBS(00)
```

Where 'command string' is of the same format as $MXE commands issued from the terminal.

The MXE will return the result of the command at address 06 followed by a 00 byte with ENDI if the 2200 sends a CBS(21) and sets CPB ready. The 2200 must take the entire result at once. If the 2200 breaks while taking the result, the rest of the result will be lost.*

* NOTE: in the bootstrap all return codes are one byte long followed by a byte with ENDI on. A return code of '00' denotes no errors.

MXE COMMANDS:

In the command descriptions that follow, please use these definitions:

'psw' is a six character password containing no blanks (the default is 'MXEPSW'), and
'port designator' is a one character designator as follows:
0 = the port at which the command is being typed (this is so that the user need not know which port s/he is connected to), and
1, 2, 3, 4 are the absolute MXE physical port addresses.

All commands begin with a ONE BYTE command code. Most commands then have a sixbyte password which is the MXE password (similar in use to the 2200 system password) followed by any needed parameters in the order specified below. The user can always type help (while in MXE command mode) to obtain a list of command codes.
The command line is devided into fields. The first field is one character long and is the 'COMMAND BYTE' field; it should be used for the one byte command. The second field is the password field. it is six characters long and begins at the third column (thus allowing one blank after the command byte). When the password is typed in the password field it will not be printed on the screen. Instead, the MXE will print the numbers 0 through 5 as characters are typed. this is so that the password can be protected. After the password field is a free format field in which the user types the rest of the command. The MXE is blank insensitive so the command parameters may be typed as the user wishes with the following exceptions:

1) A blank must not be found in a baud rate specification, and
2) The string that is to be printed on all screens (command 'G') must be typed exactly as it is to be printed.

Commands:

A) Set primary user.
Format: A 'psw' 'port designator'

The port desinated becomes the new primary user in VP mode

B) Set baud rate.
Format: B 'psw' 'port designator' 'baudrate'

The baud rate specified is set at the port designated

C) Set password.
Format: C 'psw' 'newpsw'

'newpsw' becomes the new password

D) Download code.
Format: D 'psw' 'port designator'

Z-80 microcode is loaded into the MXE at the port designated. The protocall is as follows:

If the command is not legally specified the MXE will send an 'ILLEGAL COMMAND' to the port originating the command (which may also be the port designated). Otherwise, the MXE will ask the user for a confirmation of the command. If the user responds affirmativly, the MXE will output a X'00' at the designated port when it is ready to start receiving code. It will then expect to receive ten bytes of X'00' in succession followed by code in the following format:

'2-byte destination' '1-byte count (n)' 'n bytes of code'

The MXE will continue taking in strings of this form until a count of zero is received at which time the MXE will jump to the address contained in location X'4000'. No form of error detection exists in this format once the downloading has begun so the user should be very careful to not get out of synch with the MXE. Also, all other MXE activity will halt while this command is in progress. It should be noted that the string of 10 zero bytes is needed for protection against a user designating an incorrect port address or the accidental invocation of this command.

E) Analog loopback
Format: E 'psw' 'port designator'

Analog loopback is executed at the port designated. This command will cause all I/O at all ports to be temporarily suspended!

F) Digital loopback
Format: F 'psw' 'port designator'

Digital loopback is executed at the port designated. This command will cause all I/O at all ports to be temporarily suspended!

G) Print to all screens
Format: G 'message'

Print 'message' on all terminals connected to this MXE. 'message' must not run over the 78 byte limit of the command. 'message' will be inserted in the output stream of all terminals regardless of their state and a 3 second pause will be executed to allow all users to read the message.

H) Help
Format: H

A list of all $MXE commands is printed on the terminal

I) Memory test
Format: I 'psw'

All of RAM is tested in a nondestructive way. Although all port activity on the MXE will slow down somewhat, no damage will be done.

J) Restart
Format: J

This command restarts the MXE at its poweron diagnostics. This command can only be issued while in the bootstrap. The current state of the MXE will be completely reinitialized! All baud rates which have been set through $MXE command mode will be reset to their hardware defaults!!.

L) Lock
Format: L

This command locks the current baudrate of the port issueing it. No port may change the baud rate of a port which is locked. This command is a toggle. Each time it is issued the state of the baud rate lock will be reversed.

NOTE: During many of the above commands the performance of the MXE will be impaired. Generally, this consists of all the ports slowing down somewhat. But in the case of analog and didgital loopback ALL port I/O will cease until the loopback is completed. Therefor, it is absolutly imperative that all users connected to the MXE when loopback is to be performed be notified that the MXE will not be functioning smoothly.

Baud Rates:

Below is a list of baud rates available on the MXE. Some of the rates can only be set via the $MXE baud rate command and will not have an entry in the second column. Others can be set by the hardware baud rate switches. Pleas see the discussion following the table which describes how default hardware baudrates are set. When setting baudrates via the $MXE command, the baud rate should be entered exactly as found in this table.

Rate	Hardware switch setting
50	
75	
100	
110	0
134.5	1
150	2
200	3
300	4
600	5
1200	6
undefined	7
2400	8
undefined	9
4800	A
undefined	B
9600	C
19200	D
undefined	E
see note	F

NOTE: Setting all the hardware switchs to X'FF' will cause the MXE to go into diagnostic burn in mode. In this state the MXE will continuously loop on its board diagnostics and will not talk to the 2200 or allow any user input. All board errors found will be displayed on the screen.

How to set hardware baudrate defaults:

Two eight bit baudrate selection switches exist on the MXE. Each bank of eight bits is devided into two four bit groups each of which cooresponds to a specific MXE port. To find which group is used for each port hold the MXE board with the rail in your right hand and the two banks of baudrate switches facing you at the bottom left corner of the board. There will actually be three banks of switches in this corner; the two eight bit banks mentioned above and a smaller four bit bank which is used for setting the boards I/O bus address. Disregard this smaller bank for the purpose of this discussion.

The four ports on the MXE are numbered 1 to 4 with the port closest to you being 1 and the farthest one being 4. The switch banks are devided with the top right four bits cooresponding to port 1, the top left four bits cooresponding to port 2, the bottom right to port 3 and the bottom left to port four.

Once you have located the group of switches for the desired port, find the switch setting in the table above and set the switch with the most significant bit to the left and least to the right. Notice that the bits will be ON when the switch is positioned toward you (as marked on the switch). The baud rate chosen for a port must match the baud rate of the device connected to the port. If this is not the case, communication will not be possible between the device and the MXE.

Parity errors:

The MXE hardware is equiped with parity checking in its Random Access Memory (RAM). Each time a byte is read from RAM the parity on that byte is tested. If the parity bit is incorrect the MXE will perform a parity interrupt which will cause it to return to the bootstrap no matter what it is doing at the time. If it is already in the bootstrap it will restart the bootstrap. All data will be lost in this situation.

At this point the user may enter $MXE command mode and run specific diagnostics or may restart the MXE at its diagnostic bootstrap. The diagnostic bootstrap will diagnose the board completely (as it does each time the board is powered up) and list whatever error it finds on all screens. It is NOT possible to return to the code in RAM without downloading new code into the MXE first!

DOCUMENT SUMMARY

Document Id: 0037E
Document Name: MXE 2200 interface
Operator: Eric
Author: Eric

Comments: Brief

Pages to be printed 1

Notify U13 on system PAM.

Connect enable:

CBS(22)

Enables MXE connect funtion. When a terminal connects to the a port with connect enabled the MXE will alert the 2200 via a bit being set at address 02 (the bit is set one time only!).

Connect disable:

CBS(23)

Disable the connect function.

Disable enable:

CBS(24) OBS(XXXX)

Enables the MXE disconnect function. When a terminal disconnects from a port which has this function enabled, the MXE will alert the 2200 via a bit set at address 02 (the bit is set one time only!). Additionally, if XXXX is not equal to hex(FFFF) then the connected terminal will be disconnected by the MXE in XXXX seconds. If a previous command set a disconnect time then this new time will replace (starting the count at 0). If XXXX is hex(FFFF) then any current count down is disabled and the terminal is not disconnected. The timeout is a ONE TIME countdown and will not have any effect once the countdown is terminated (by disconnection or being disabled). when a terminal disconnects all unfinished countdowns are disabled

Disconnect disable:

CBS(25)

The above function is disabled and all countdowns are disabled.

DOCUMENT SUMMARY

Document Id: 0039E
Document Name: Remote Screen Dump (RSD)
Operator: Eric
Author: Eric

Comments: Preliminary

Pages to be printed 4

Notify U13 on system PAM.

From: Eric Wilson
Subject: MXE Data Transfer Facility
Date: 03/28/83 (revision of 06/15/82 memo)
Pages: 4 (including cover)

- -

Your comments are welcome and should be directed to Eric Wilson at X7192, M/S 1389A in Lowell Tower II.

c.c.

	(1) T - MXE (normal state)	(2) T-MXE (DT)		(3) MXE-T (DT)
F0	Edit key		D0	
F1		Escape	D1	
F2		Start Data	D2	
F3	Request Data Transfer		D3	Start Data Transfer
F4		End Data Transfer	D4	Abort Data Transfer
F5			D5	
F6			D6	
F7			D7	
F8	CRT go	CRT go	D8	Send
F9	PRT go	PRT go	D9	
FA	CRT stop	CRT stop	DA	Wait
FB	PRT stop	PRT stop	DB	
FC	Transparency		DC	
FD	ENDI/ATOM		DD	
FE	Dead key		DE	
FF			DF	

During a Data Transfer (DT) the MXE will use the codes in column 3 above to control the flow of bytes from the connected device. Column 2 contains the allowable commands from the device to the MXE during a DT. All the commands in column 2 must be preceeded by escape byte 'F1'. 'F1' in the normal data stream must also be escaped. Thus, a data byte of 'F1' would be sent as 'F1''F1'. An end of DT command would be sent as 'F1''F4'. Commands from the MXE to the device must be escaped by escape byte 'FB' which is the normal MXE to Terminal escape byte. 'FB' in the data stream will be converted to 'FB''D0' to be compatible with older terminal formats.

Data flow is allowed in both directions (to and from the connected device) to allow for the transmission of error checking codes in the reverse direction.

NOTE: A device MUST issue an'E5' to the MXE immediatly following EVERY reset and restart. This is used by the MXE as an indication that the device supports Data Transfer. The MXE will NOT allow DT with a device which has not issued an 'E5' since the last reset or restart. (This is to prevent a DT from being attempted with a device which does not support it).

General Command Sequence:

In the following description 'T-MXE' means 'From the device to the MXE', 'MXE-2200' is 'From the MXE to the 2200', and so on.

Device initiation:

T-MXE: 'F3''ID'*
MXE-2200: Set DT bit at address 02.
MXE-2200: Pass 'ID' at address 06.

A request has now been made to the 2200 for the desired Data Transfer. The MXE will continue in its normal mode until the 2200 initiates the DT as follows. (in the case of a terminal requesting a Remote Screen Dump, the terminal will also remain in its normal state until the 2200 initiates the RSD. This is to prevent the MXE and terminal from hanging while waiting for the 2200 to get ready. The 2200 may not be able to honor the terminals request at all!):

2200-MXE: Command at address 06 - pass 'ID' followed by any needed parameters for the requested DT.

The 2200 is now ready to proceed with the transfer and will wait for the devices to proceed.

MXE-T: 'FB''F3''ID''parameters'

Both the device and the MXE prepare for the DT. When the MXE is ready:

MXE-T: 'FB''F8'

The device should now begin the transfer. As needed the MXE will send the following Go and Wait codes to the terminal to control the flow of data once the transfer has begun:

Go: 'FB''F8'
Wait: 'FB''FA'

Five conditions may terminate a DT. They are as follows:

1) Normal termination: T-MXE: 'F1''F4'
MXE-2200: Set DT bit at address 02
MXE-T: 'FB''F8'

2) 2200 Abort: 2200-MXE: Abort DT command at address 06
MXE-T: 'FB''F4'
MXE-T: 'FB''F8'

* See section on ID for a complete description

3)	Reset Abort:	T-MXE:	'F1''12'
		MXE-2200:	Set DT bit at address 02
		MXE-2200:	Set Reset bit at address 02
		MXE-T:	Normal terminal reset procedure
4)	Device disconnect:	MXE-2200:	Set DT bit at address 02
		MXE-2200:	Set Disconnect bit at address 02
5)	MXE Abort:	MXE-T:	'FB''F4'
		MXE-T:	'FB''F8'
		MXE-2200:	Set DT bit at address 02

It should be noted that the device connected to the MXE does not have to be a terminal. It could be some other device (EX: Wangwriter) which usually looks like a terminal to the MXE but may use the MXE for Data Transfer other than a Remote Screen Dump. For this reason, the above protocall has been kept as general as possible. The MXE never has to know the format of the data being transfered nor the number of bytes.

ID:

A one byte ID is used to designate which type of Data Transfer is being requested (by the device) and which type is being initiated (by the 2200). Currently, only two ID's exist. An ID of 01 specifies Remote Screen Dump and an ID of 02 specifies File Transfer. As additional uses for the Data Transfer facility of the MXE are generated, new IDs will be assigned. In any case, the MXE does not care which transfer is being done so the ID is not important to it at this time.

Following the ID, the MXE will send to the terminal a list of parameters (four bytes in the case of RSD). For RSD this parameter list is a specification of the part of the screen which is to be dumped. The first two bytes specify the upper left corner of the area while the second two bytes specify the lower right corner. The first byte of each group is the column number (from 0 to 79 for a 2336DW terminal) and the second byte is the row number (from 0 to 25 for a 2336DW terminal). The MXE receives the parameters from the 2200 when it issues a DT command.

Compression (RSD):

Data compression has been defined for an RSD. The byte 'EC' followed by a count of the number of bytes being compressed followed by the byte being compressed will be used. To send an 'EC' the terminal must send 'EC''EC'. Counts of 'EC' (decimal 236) and 'F1' (decimal 241) are not allowed (ie. a string of 236 or 241 compressible bytes must be broken into two compressions).

'EC' 'count' 'compressed byte'

DOCUMENT SUMMARY

Document Id: 0043E
Document Name: MXE summary sheet
Operator: Eric
Author: Eric

Comments:

Pages to be printed 1

Notify U13 on system PAM.

2236MXE Terminal Multiplexer

The 2236MXE terminal multiplexer is a Z80 microprocessor based device controller with the capability of controlling upto four 2236D, 2236DE, 2236DW, 2336DE, and 2336DW terminals in any combination.
The following is a list of 2236MXE features:

1) 2236MXE operating code is downloaded into the controller at configuration time to allow various applications to be performed by the same board in a variety of operating environments,

2) A time of day clock enables the 2200 system, through the 2236MXE, to provide the Basic TIME and DATE functions (in both Basic-2 and Basic-3),

3) In addition to local terminals, the 2236MXE has the ability to support remote terminals through the use of an RS232 compatible modem (such as the wang WA3451 full duplex modem or similar),

4) A special $MXE command mode allows the user to communicate directly with the 2236MXE. This allows the changing of buadrates, modem diagnostics, board diagnostics, and so on (see below),

5) Terminal port baud rates are both hardware and software selectable. At powerup the baud rate switches are read to determine the default powerup baudrates. The baudrates may be changed at any time through the use of the baud rate command in $MXE command mode.

6) An extensive set of diagnostics is run at powerup to completely test all board components and provide these results to the customer engineer in case of faults. Many of these diagnostics can be rerun through $MXE command mode,

7) The wang WA3451 modem can be tested by the 2236MXE through $MXE command mode,

Future Enhancements

An INPUT SCREEN statement is currently being implemented to allow the user to input an exact image of the users screen through basic. This function requires the use of the 2236MXE.

An Asynchronous software package similar to that on the 2227B TC board will be implemented soon.

A file transfer function will be added to the 2236MXE to allow high speed transfer of large amounts of data (such as WP documents) through the 2236MXE in background while normal terminal processing continues.

DOCUMENT SUMMARY

Document Id: 0046E
Document Name: ATOMS (copy)
Operator: ROGER KIRK
Author: ROGER KIRK

Comments:

Pages to be printed 1

Notify U13 on system PAM.

	8	9	A	B	C	D	E	F
0	LIST	TRACE	PRINT	STEP	FN	(a)SIN(	LS=	LINPUT
1	CLEAR	LET	LOAD	THEN	ABS(	(a)COS(	ALL	VER(
2	RUN	FIX(	REM	TO	SQR(	HEX(	PACK	ELSE
3	RENUMBER	DIM	RESTORE	BEG	COS(	STR(	CLOSE	SPACE
4	CONTINUE	ON	PLOT	OPEN	EXP(	ATN(	INIT	ROUND
5	SAVE	STOP	SELECT	(s)CI	INT(	LEN(	HEX	ATC
6	LIMITS	END	COM	(s)R	LOG(	RE	UNPACK	HEXOF(
7	COPY	DATA	PRINTUSING	(s)D	SIN(	(s)#	BOOL	MAX(
8	KEYIN	READ	MAT	(s)CO	SGN(	%(image)	ADD	MIN(
9	DSKIP	INPUT	REWIND	LGT(	RND(	(s)P	ROTATE	MOD(
A	AND	GOSUB	SKIP	OFF	TAN(	BT	$(gio)	resv
B	OR	RETURN	BACKSPACE	DBACKSPACE	ARC	(s)G	ERROR	resv
C	XOR	GOTO	SCRATCH	VERIFY	#PI	VAL(	ERR	resv
D	TEMP	NEXT	MOVE	DA	TAB(	NUM(	DAC	resv
E	DISK	FOR	CONVERT	BA	DEFFN	BIN(	DSC	resv
F	TAPE	IF	(s)PLOT	DC	(a)TAN(	POS(	SUB	P.L.N.

(s)=select

(a)=arc

resv=reserved

P.L.N.=Packed Line #.

COMPANY CONFIDENTIAL

Report on Wang Deutschland X.25 Project
(Current Technical Problems)
13 January 1983

By: Eric Wilson
To: Neeraj Sen and Bruce Patterson

c.c. Heinz Jurack, Peter Thornton

COMPANY CONFIDENTIAL

TABLE OF CONTENTS

This report contains a description of the Wang Deutschland X.25 network project along with a detailed discussion of current problems and possible solutions. It is the result of a four day visit to Wang Deutschland facilities in Frankfurt and Hamburg Germany the week of December 6 1982.

All comments pertinent to this report should be directed to:
Eric Wilson at Mail Stop 1389A in Tower I Lowell.

1 SYSTEM DESCRIPTION

Wang Deutschland GMBH, Transfer Data Test GMBH (TDT), BDB, and Shell Oil of Germany have undertaken the task of establishing a large distributed network within Germany for Shell Oil and its retail supply network. The purpose of the network is to facilitate highspeed, online communications between various members of a distributed data base and shared resources. The main intent of the project is to provide links between remote offices and their regional computing (and controlling) centers as inexpensively as possible while retaining full on line capabilities.

It should be noted that British Petroleum (BP) has also expressed interest in establishing a similar network and that Wang Canada in Burnaby has expressed concern over problems they are experiencing with a similar network.

1.0.0 System Geography:

Ten central computing and controlling centers located throughout Germany provide regional computing facilities for their respective branch offices with Wang 2200 multiprocessing equipment. Each branch office is connected to its respective central computing center over the Deutsche Bundespost X.25 public packet switching network. The central computing centers in turn communicate over the X.25 network thereby providing communication between all points in the network. (It should be noted that although communication is possible between ALL members of the network, each branch office communicates directly with its regional computing center only)

1.0.1 System Interfaces

Every device connected to the X.25 network is treated as a 'Generic Device'. No distinction is made between CPU's and terminals by the network. No master slave protocol exists at the system packet level thereby creating a true 'packet delivery system'. The network is therefore transparent to the connected devices up to the packet level and transparent to all application software.

At each end of a link the X.25 network connection is interfaced to the Wang equipment by means of an X.25 to RS232C converter (MPAC) manufactured by TDT (see figure). The MPAC converter performs all conversion functions including packet manipulation and error handling.

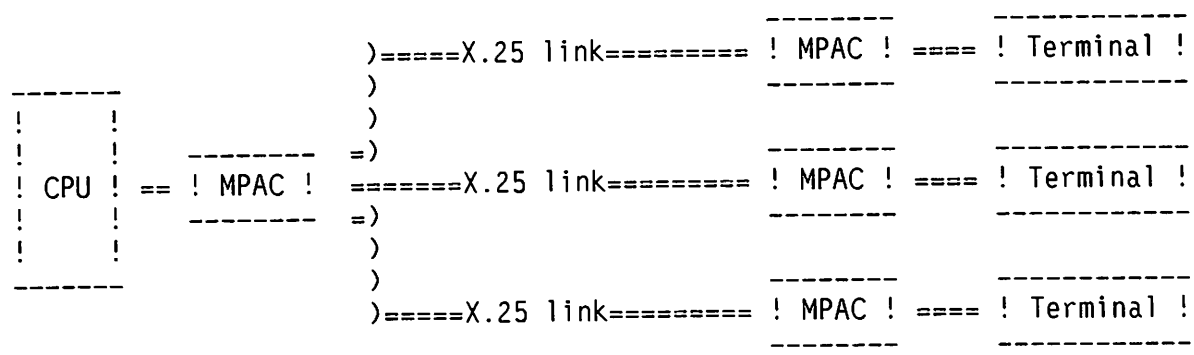


Figure 1: Typical link configuration

2 TECHNICAL PROBLEM DESCRIPTION

Unfortunately, the X.25 connection is currently not ABSOLUTELY transparent to the devices utilizing it. When the device communication protocols for these devices were established the packet switching environment did not exist. Adjustments need therefore be made to the protocols to maximize there functionality and efficiency in this new environment.

2.0.0 Cost

Charges for use of the X.25 connections are based on connection time and packet use. A flat charge is levied per minute of connection time. A flat charge per packet, regardless of the number of data bytes in the packet is also charged. (i.e. a packet containing one data byte costs the same as one containing 128 data bytes) It is therefore desirable to PACK each packet as much as possible and to minimize traffic on the network to only that which is absolutely necessary.

2.0.1 Echo

Currently, each byte entered at the keyboard is transmitted over the X.25 network to the CPU. The byte is then processed by either the CPU or the terminal controller (MXD or MXE) and one of three actions is taken:

- The byte is DIRECTLY ECHOED to the CRT at the cost of one X.25 packet (two for the round trip), or
- The byte is TRANSLATED into some number of bytes which are then sent to the CRT as a group (examples: Text Atoms, end of line characters) at the cost of one X.25 packet (two for the round trip), or
- The byte is passed to the 2200 CPU which then takes appropriate action (examples: end input mode, recall line).

As cost is of major importance it is highly desirable to eliminate unnecessary packet traffic between the terminal and the CPU. Packet loading (number of data bytes per packet) and echo are of importance in this consideration.

2.0.2 Performance

Internal to the Deutsche Bundespost X.25 packet switching network the packets are transmitted at a rate of 64Kbps. However, the actual links between the central X.25 processors and the devices are usually at rates around 4800 bps to 9600 bps. These lower rates being the limiting ones it can be plainly seen that a substantial delay must be experienced by each packet. Thus, bytes which are remotely echoed by the terminal controller will take longer to reach the CRT than packets locally echoed. A local echo function (where possible) would greatly increase the performance of the system.

2.0.3 Flow Control

The flow control currently used by locally attached terminals is not suited to the packet environment. When the terminal finds itself lagging behind the data stream at the RS232C connection it sends out a STOP code which causes the terminal controller to stop transmissions until it receives a GO code. Additionally, two data streams (the CRT and PRT) are multiplexed on this channel through the use of two STOP/GO code sets. In the X.25 environment these codes are delayed such that by the time the terminal controller stops transmitting the terminal is already overflowed causing an error condition. To prevent this from happening, the MPAC has a flow control protocol of its own. The MPAC on the terminal end of the link reacts to the flow control commands from the terminal as the terminal controller does. However, the MPAC on the terminal controller side uses the RS232C Clear to Send (CTS) signal to the terminal controller to stop the controller sending rather than the flow control codes from the terminal (some flow control bytes are passed through the X.25 link to keep the controller sending to the proper device). This is not supported by the current release of MXE terminal controller software as it is not used with standard Wang terminals. CTS flow control must be supported if the MXE is to be used with the MPAC.

3 POSSIBLE SOLUTIONS

The following three solutions are those which I feel are most appropriate and workable given the current state of all system components as they now stand.

3.0.0 SOLUTION 1: MXE Flow control

The reason the MXE is not currently configured to allow CTS to be used for flow control is three fold.

In the first place, the standard Wang terminal cable does not fully support all standard RS232C signals. Many standard signals are missing and thoses which do exist are not always used in the standard way. CTS is always tied to Data Set Ready (DSR) in the cable thereby making it impossible for the terminal to manipulate this signal. Thus, it has not been necessary for the terminal controller connected to a Wang terminal to respond to CTS as flow control.
Secondly, as two data streams (CRT and Printer) are multiplexed over the terminal cable it is inefficient for the terminal to use CTS to stop the flow. Both streams would be halted when only one needed to be.
Lastly, the terminal cable does not provide Data Carrier Detect (DCD) to the MXE. Since the standard use of CTS as flow control (in full duplex systems using standard ZILOG interpretations) is CTS ANDED with DCD the absence of this signal would cause the transmission to be halted at ALL times.

It is therefore clear that in order to use CTS as flow control to the MXE the device attached to the MXE must perform certain functions:

- Instead of a standard Wang terminal cable the device must be connected using a cable which fully supports all (or most) RS232C signals. The standard Wang TC cable fulfills this need.
- The device must supply DCD and CTS as VALID control signals.
- The device must pass on the flow control codes from the terminal for the CRT and Printer so that the CTS signal is used only as a secondary flow control mechanism. (i.e. the CTS signal is used to prevent the MPAC from being overflowed while the normal terminal flow control codes are used a level above to control the two streams to the terminal).

The MPAC fulfills all the above criteria. With some minor modification to the MXE software it can be capable of detecting devices which supply CTS as flow control and respond accordingly.

3.0.1 SOLUTION 2: MXE no echo configuration

The echoing of PRINTABLE ASCII characters and backspace would be handled by the MPAC in this configuration. All bytes would be sent to the MXE as is currently done (with somewhat better packing of the packets) but the MXE would send no bytes to the terminal except in the following circumstances:

- TEXT ATOM key depressed, or
- Special Function key depressed requiring printing to the screen, or
- Editing key depressed such as insert or delete mandating screen updating.

This procedure would minimize the delay time required for each printable ASCII character to reach the screen and would greatly reduce the number of packets required. Unfortunately, some major drawbacks exist with this solution:

- No wraparound of the line could be allowed as this would cause a race condition between the MPAC and the MXE. At the end of a line the terminal controller repositions the cursor to the beginning of the next line by means of a cursor down control byte. It is highly likely that by the time this control byte is received by the terminal the MPAC would have already echoed the next byte causing the data on the screen to be in error. It is not possible for the MPAC to generate the proper end of line sequences without a protocol between it and the MXE being established. Thus, all linputs must be limited to one line (i.e. may not cross line boundary).
- A backspace off the left side of an input field should not be echoed. The MPAC will echo this backspace.
- When a special function key is pressed the MPAC must delay the echo of the next printable ASCII byte until the MXE has time to take any action necessitated by the function key. This must be done to prevent events from occurring in the wrong order. (such as a text atom being printed two bytes after it is depressed).

3.0.2 SOLUTION 3: Shifting of Responsibility

This solution addresses the problems unattended to by the above solution by means of a protocol between the MXE and the MPAC. The MXE would inform the MPAC, through a protocol addition, of the information it needs to control the keyboard and CRT handling more completely. Special function keys and TEXT ATOMS would still be handled by the MXE but all other editing functions would

be performed by the MPAC directly. The number of packets needed would be further decreased while some performance efficiency would be realized. Unfortunately, the MPAC would no longer be a transparent part of the link. The code used in the MPAC would need to be specific to the MXE application and would probably require substantial changes. The next solution deals with these issues.

3.0.3 SOLUTION 4: PC screen handler

With a change of hardware this solution would prove the most effective and easiest to maintain. Instead of a DE or DW terminal a stripped down Professional Computer (PC) would be employed. Two possibilities exist:

- A stripped down, low cost version of the PC with one terminal attached would act as a remote MXE terminal controller. Most of the functions currently performed by the MXE would be done by the PC. No control communication between the MPAC and the MXE or PC would be needed thereby keeping the MPAC a transparent part of the link. Data could be highly packed as the PC would communicate with the MXE only when absolutely necessary. This solution would realize the greatest reduction in packets employed.

- Similar to above this solution would use the extra RS232C port on the PC to control a second terminal from the same PC. Both terminals would use the same X.25 connection thereby realizing further savings in connection time and fewer packets. The data stream would be multiplexed at the MXE through one port (i.e. two MXE logical ports would use one MXE physical port) transparent to both 2200 MVP Basic and the MPAC's.

3.0.4 Generally speaking

In all the above solutions some method of allowing some ports to continue to work in their current 'local terminal support mode' while others on the same MXE work in 'remote echo off mode' needs to be formulated. An addition to MXE command mode could support these new functions. Additionally, the MPAC might be programmed to automatically send a control sequence to the MXE on connection for automatic configuration transparent to the user.

4 Suggested solution

Solution 1 must be done in all cases. Solution 2 is a quick fix solution; One parameter in MXE command mode could invoke this. Solution 3 is actually a superset of 2 and could be treated as a parameter when selecting the mode. The PC solution, although the most desirable, would require a much longer developement effort and must wait until appropriate hardware is available. It is quite possible that an X.25 link may be added to the PC thereby making it unnecessary to use only one MPAC unit (at the MXE side). Such an addition would make this solution most desirable and easily implementable.

For the immediate future I recommend that we implement the the CTS solution and solution 2 with the possibility of adding solution 3 or perhaps the PC solution later. It would require two days to change the code in the MXE to support the 1 and 2. A pre-release version for testing could be sent with subsequent adjustments being made later with not much time commitment required. Solution 3 would require more iterations. The quickest and easiest method would be on-site implementation in coordination with TDT.

FROM: Eric Wilson
SUBJECT: NML Trip Plans
DATE: 4/4/83

Distribution:

Neeraj Sen	
Rich Racicot	
Bruce Patterson	
Jerry Sevigney	
John Deutsch	MS 14A7A
Gene Mantoni	MS 14A7A
Neil Aronson	MS 14A7A

The following list contains those problems reported by NML and some other users which I consider the main thrust of my investigations at NML and Dayton*:

1) Terminal port lockup on 2236 MXE terminal controllers.

2) Extraneous HEX(F0) (CANCEL key) somehow received by the MXE (or Option W) when printing to a remote terminal printer but not generated by the terminal (thereby aborting the print job).

3) Garbage generated by a disconnecting modem connected to a terminal thereby locking up the terminal necessitating the cycling of power to the terminal to regain terminal functionality.

* See Northwestern Mutual Life letter of 3 March 83 from Mark A. Prange to John Deutsch.

In order to investigate these problems I plan to do the following before and during my upcoming trip to the Milwaulkee and Dayton sites (my investigation will certainly not be limited to this memo whose primary purpose is to promote commentary):

1) Before the trip I will arrange to have the power sources for all systems at both sites monitored for line noise and voltage tolerance.

2) Both cites will have selected systems tested for static tolerance.

3) Those systems which seem exceptionally problematic will have these additional tests performed:

   All I/O bus voltages will be checked with an eye toward adjustment where needed.
   Where possible, an attempt shall be made to monitor I/O bus characteristics during various system conditions.

4) I shall monitor the RS 232 terminal connections to help determine the state of the system just before failure.

5) All test results and observations will be meticulously recorded for future study.

6) I shall carry with me various software tools to facilitate software testing.

7) If possible, I will arrange for a particularly inoperative system to be brought back to lowell to try to reproduce system failures in a fully equiped R&D environment. (Customer Engineering tells me that a swap with the user for a bad system may be possible)

8) I will have a developement system in Lowell available for my use over TC lines should I require software assistance apart from that which I take with me.

I expect to receive assistance from Customer Engineering in the form of a Tech qualified to work with 2200 LVPs at both a system and basic electronic level. The Tech should come equiped with a static tester and be qualified to use one.

FROM: Eric Wilson
SUBJECT: Milwaukee-Dayton trip report
DATE: 5/6/83

Distribution:

Neeraj Sen	
Rich Racicot	
Bruce Patterson	
Jerry Sevigney	
John Deutsch	MS 14A7A
Gene Mantoni	MS 14A7A
Neil Aronson	MS 14A7A
Michael Riley	MS 8236A

This report is the result of a trip to Northwestern Mutual Life (NML), Milwaukee, Wisconsin and C.H.Dean, Dayton, Ohio made by myself and Mike Riley (Customer Support) on 21 April 1983.

Purpose of the trip

The purpose of the trip was to investigate critical system problems reported at each of the sites. The problems reported were:

1) Terminal port lockup on 2236 MXE controllers.

2) Extraneous HEX(F0) (Cancel key) being received into the basic program ouputting a WP document to a remote system via the Remote Control Maintenance package (RCM). This problem caused the print job to be prematurely aborted. (NML only)

3) Garbage generated by a disconnecting modem connected to a terminal thereby locking up the terminal necessitating the cycling of power to the terminal to regain terminal functionality. (NML only)

Other problems

In addition to the above both sites presented additional problems which they considered less critical. In the problem status section each problem or topic of interest is discussed singly.

System configuration

NML: Four LVP-C cpus with one or two 2236 MXE controllers and one 2228D TC controller board.
2236DE, 2236DW and 2336DW terminals.
Other unrelated 2200 equipment.
Port one on each system is local while all other ports are remote through Racal Vadic modems (equivalent to Wang WA3451) mounted on large racks.
Each 2228D connects remotly to an IBM host.
All Wang cpus are located in the same room in two single files with many other pieces of communications and processing equipment.

C.H.Dean:
One 2200 MVP cpu with two 2236 MXE controllers plus printer controller plus disk controller.
One 2200 MVP cpu with one 2236 MXE and one 2236 MXD plus printer controller plus disk controller.
2236DE and 2236DW terminals.
Other unrelated 2200 equipment.
Both 2200 MVP cpus are located side by side near system printers along an office wall in a room with static mats.

Specific problems and their status

1) Terminal port lockup on 2236 MXE terminal controllers. (NML and C. H. Dean)

Symptoms: A port on the MXE locks up for no apparent reason. It is sometimes possible to regain port functionality through the use of MXE port diagnostics from another port on the same MXE although more often than not the system must be rebooted to restore the port. Additionally, the problem tends to stay with the system rather then the MXE when MXE controllers are swapped. Frequency of failure varies widely from system to system with some systems failing as often as every few hours while others may not fail for weeks (if at all).

Investigation: For two weeks prior to the trip the power lines at both sites were monitored for abnormalities. Large spikes were detected at NML but these do not seem to have any connection with system failures. Both sites showed a large variance in line voltage, again with no apparent coerelation.
Static testing was performed at the Dayton site (NML has too much equipement in the vicinity of the systems to perform static testing). The system tested out at 2,500 volts with the 2236DW terminals failing before any other system component.
Unfortunatly, the system log at NML (which I requested be kept starting 2 months prior to the trip) was not kept properly and was therfor of no use in these investigations. At Dayton a better log was kept with copies of the screens attached to the locked up port at the time of the failure whenever possible. Unfortunatly, many of the failures occured at night.
While at NML no system failures occured. Also, the frequency of failures has recently dropped way down. It has become standard practice to reboot all systems every morning and this may be a contributing factor to the decline in system failure frequency. It is also possible that a recent software modification could have contributed to this decline.
While at Dayton we were able to produce the port lockup problem consistently. Sufficient information was collected to cause the problem to occur in lowell.

Problem resolution: One cause of the port lockup problem was traced to the $DISCONNECT function. If a disconnecting port was in control of more than one partition at the time of disconnection AND the highest numbered partition was a background partition then the system would not command the MXE port to allow a reconnection thereby locking the port. THIS PROBLEM HAS BEEN RECTIFIED in the latest version of the operating system which went to Q&A on 4 MAY 1983 (Rel 2.5.2)
HOWEVER! It is not clear that this problem is completely solved! It is highly possible that the $disconnect bug was only a contributor and that some other bug still exists. This bug could be a software bug or it could be a hardware bug (such as sensitivity to static).

I have commited Wang to the future investigation of this problem until a satisfactory solution is implemented.

2) Extraneous HEX(FO) (Cancel key) problem (NML only)

Symptoms: During the printing of a document to a remote system using the Remote Control Maintenence package over a TC line a HEX(FO) without ENDI (i.e. not a control byte) is received by the basic program in the host. A HEX(FO) is interpreted as a cancel key and therefor aborts the print job. No key was pressed on the terminal to generate this byte ans the source is unknown.

Investigation: This problem could not be duplicated while I was at NML. NML no longer considers this a problem so no further action will be taken on this problem.

I made no commitments to further investigate of this issue.

3) Garbage generated by a disconnecting modem causing terminal lockup (NML only)

Symptoms: A terminal connected to a modem which has just disconnected from a system will lock up when the next connection is made. The terminal must be powered off and on to get it working again.

Investigation: This problem is repeatable and I saw it happen 3 times during 20 trials. The phone line is dropped by switching the modem to the off-line position. Either the modem sends a few garbage bytes to the terminal or sets the RS232 signals into a state not expected by the terminal (it is not clear which) causing the terminal to occasional hang.

Problem resolution: Unfortunatly, Wang 2236 and 2336 series terminals are sensitive to garbage on the line. I have had this problem in the lab not only with modems but when changing cables or testing code which may send illegal bytes to the terminal.
I see no solution apart from rewriting the 2236 and 2336 terminal code from scratch and possibly redesigning the RS232 interface. I do not think this is necessary as NML considers this a nuisance but not a major problem and no other users have considered this enough of a problem to report it (or no other user has experienced this problem).

No commitment was made by me to provide a solution to this problem. NML was informed (by me) that there would most likely be no action on this issue by Wang.

4) Printer defaults: NML some time ago requested special proms for their 2233, 2235, DW20 and 2281 printers to change the page length defaults to 51 ($8^1/_2$ inches) lines from the current 66 to match the format used by NML on all their forms. These proms were promised by Joe Sapienza but were never created. It has been many months and no proms have been provided to NML nor has any contact with various Wang reperesentatives (including Joe who has repeatedly promised that they are forthcoming) produced results.

Action taken: This problem has been refered to Rich Racicot. No commitment has been made by me apart from refering this issue to the proper channels within Wang.

5) "Chinese printer problem": NML has experienced a problem with some of their terminal printers which they have chosen to name the "Chinese printer problem". The printer will occasionaly begin printing garbage (It seems to lose track of where the print wheel is therby transposing the character set) instead of the proper document until the next printer reset is sent which brings the proper character set back.
This problem is seems to be identical to one often experienced in-house which is caused by a faulty daisey wheel position sensor.

Solution: Replace the daisey wheel position sensor.

6) 7 bit terminal protocall request: Many networks currently in use do not support the 8 bit data byte size used by Wang 2236 and 2336 terminals. This is a problem for NML as they would like to allow their network users to tie into their network through other existing networks. They are therfor quite interested in Wang switching to a 7 bit protocall.

Action taken: I was very firm in telling NML that we currently have no plans to switch to a 7 bit protocall but that we have discussed this issue and would be sure to let them know if we decide to support it. Absolutly no commitment was made on this issue and this was made very clear to Mark Prange.

7) A list of 2200 Word Processing bugs was given to me by Darrell Fulton at C.H.Dean. I have passed this list on to Rich Amico who has promised to attend to them and contact both Darrell and myself.

The only commitment made by me to Darrell was to forward his bug list to the proper persons.

Other points

Jim Jefferies, Bob Hayden, Blane Woodard and Mark Prange all made it very clear to me that the greates problem NML has with Wang is our reluctance to say NO to NML when we are unwilling or unable to fill an NML request. They would much rather have us say NO than have us make promises we cannot fulfill. As Jim put it, "They may no like it but it at least helps them find alternate solutions before it is too late.".

The technical support provided by Wang Customer Engineers Kevin Knepp and Ron Brockman at the Dayton site was exceptional. They were both willing and eager to do anything that was needed (including working odd hours) to aid me in my investigations.

Contacts:

Northwestern Mutual Life, Milwaukee, Wisconsin:

Mark Prange	Systems coordinator group leader, Teleprocessing
Bob Hayden	Systems Analyst, Teleprocessing
Blane Woodard	Systems Analyst, Teleprocessing

Wang Labarotories, Inc, Brookfield, Wisconsin:

Jim Jefferies	Senior Systems Consultant, Wang Brookfield

C.H.Dean & Associates, Inc., Dayton, Ohio:

Darrell N. Fulton	Director, Information Systems

Customer Engineering, Cinicinnati, Ohio:

Kevin Knepp	Customer Engineering Reperesentative
Ron Brockman	District Technical Specialist

DOCUMENT SUMMARY

Document Id: 0066E
Document Name: MXE RS-232 Pinout
Operator: Eric
Author: Eric

Comments:

Pages to be printed 1

Notify U13 on system PAM.

MXE/SVP Option W RS-232c Connection Pinout

Pin #	EIA	CCITT	Name	Signal Source
1	AA	101	Protective Ground	
2	BA	103	Transmit Data	DTE
3	BB	104	Receive Data	DCE
4	CA	105	Request to Send	DTE
5	CB	106	Clear to Send	DCE
6	CC	107	Data Set Ready	DCE
7	AB	102	Signal Ground	
8	CF	109	Received Line Signal Detector*	DCE
9			Unconnected	
10			Unconnected	
11			Unconnected	
12			Unconnected	
13			Unconnected	
14			Unconnected	
15			Unconnected	
16			Unconnected	
17			Unconnected	
18			Remote Analog Loopback*	DTE
19			Unconnected	
20	CD	108.2	Data Terminal Ready*	DTE
21			Remote Digital Loopback*	DTE
22	CE	125	Ring Indicator*	DCE
23	CH/CI	111/112	Data Signal Rate Selector*	DCE/DTE
24			Unconnected	
25			Unconnected	

* This pin is not connected on port 1 of an SVP Option W (Ports 2 and 3 utilize these signals)

To: Melanie Kempton
From: Eric Wilson
Date: 1 November '83
Subject: Estimation of work to add PC disk access to MXE

Distribution: Bruce Patterson
Jerry Sevigny
Rich Racicot

These additions need to be added to the protocol between the MXE and the PC:

- Disk data transfer channel in both directions
- Disk channel flow control in both directions
- Disk channel selection in both directions
- Disk channel control commands in both directions
- Keyboard channel selection from PC

The following restrictions, changes and functional modifications should be made to facility implementation:

- Either drop Remote Screen Dump or change the way it functions. If RSD is kept then it is a mutually exclusive channel (as is the case with all current channels). We may want to rewrite the way RSD is handeled in the protocol to allow it to be treated as just another channel.
- Clean out the protocol for the purpose of talking to the PC. There exist many leftovers in the protocol due to "History" (how I hate that word!) which keep us from easily adding functionality. For example, many TEXT ATOMS have never been used on any terminal and will certainly not be used on the PC; let's get rid of them.

These additions need to be added to the interface between the 2200 and the MXE:

- Add address 06 commands (command address) to control disk channel flow. A set of commands could be passed on the disk channel transparent to other functions.
- A different address must be used to transfer data between the 2200 and the MXE. Unfortunatly, no unused, bidirectional address exists thereby necessitating the multiplexing of another address (perhaps address 03 which is currently used for RSD and TC).

Time requirements:

If someone other than myself is to do the MXE modifications it could take 2 months or so to get up to speed on the MXE. Someone with microcodeing experience would get ther a bit faster. The MXE has gone through so many changes that it is currently very delicately balanced. Many changes could bring on timing problems which could, as we've seen from past experience, cause large delays in the deadline.

Assuming RSD is dropped and not including time to change the MVP OS I expect the changes to require:

PC-MXE protocol:	10 - 15 persondays
MXE-2200 interface:	8 - 10 persondays

This ESTIMATE does not include QA, Beta-site or design time of which design time has proven to be the longest.

DOCUMENT SUMMARY

Document Id: 0074E
Document Name: Remote editor proposal
Operator: Eric
Author: Eric

Comments:

Pages to be printed 8

Notify U13 on system PAM.

Subject: Remote Terminal Editor for the MXE and 2336 terminal
From : Eric Wilson
Date : 13 MAY 1983
Distribution:

Neeraj Sen	MS 1489
Bruce Patterson	MS 1389A
Jerry Sevigney	MS 1389A
Rich Racicot	MS 1489
John Deutsch	MS 14A7A
Peter Thornton	MS 13A3B
Tim Sloane	MS 13A3B

All comments and questions concerning this document should be directed to:
Eric Wilson, MS 1489, Tower II ext 7192

This document describes the proposal for a Remote Terminal Editor for use with the 2236MXE terminal controller (Some other terminal may be used instead depending on the terminals capabilities). The editor would be initially implemented on the 2336DW terminal and could be added to the PC terminal emulator package and other keyboard entry devices of the future. This proposal provides a long term solution to the X.25 packet switching network problem in that it solves the network delay problem and economizes the use of packets over the network. The savings provided by this proposed system would not be limited to X.25 packet switching networks. On the contrary, this system could be used with any type of communications connection between the terminal and controller including packet networks, leased line, telephone lines, and direct connections.

The Remote Editor is made possible through an extension to the existing Terminal-Terminal Controller protocol. This extension is outlined in the following pages. It is suggested that the reader refer to the "2200 Terminal-Terminal Controller Communication Protocol" document dated May 1983 by Eric Wilson which describes the existing protocol in depth. Familiarity with the existing protocol would be helpful to the reader since this document describes the proposed extension only.

NOTE: This proposal does not cover every detail in full. The structure of the solution is presented here and those details which do not effect the structure as a whole have been left to future study. The reader should not be alarmed by this as all the major structural specifications are contained herein.

Functional description:

The idea of the Remote Editor is to allow the terminal to perform as much of the screen editing as possible thereby minimizing traffic over the communications link and speeding up the editing process. Unfortunately, not all editing functions can be independently performed by the Remote Editor since interaction with the 2200 operating system is needed for some keys. Editing functions to be performed by the Remote Editor are insert, delete, erase, cursor movement, backspace, Text Atoms and normal character entry. All special function keys including special functions 0 to 31, Prev and Next screen, Recall, Tab, GL and return must be handled by the 2200.

Current editing system:

All editing is currently done by the MXE with occasional 2200 interaction where needed (all cases stated in the last section). The MXE is responsible for screen updates as each key is pressed on the keyboard. When a key is pressed which the MXE can not handle independently it signals the 2200 which then instructs the MXE. In most cases the 2200 does not care what is in the edit buffer since most keys are independent entities (ie the action taken in response to a key is not dependant on anything that came before it). In the cases of Recall and return the 2200 will need to read all or part of the edit buffer in order to determine the next course of action.

Suggested system changes:

The terminal would be commanded by the MXE to execute a line request of with a certain size and initial conditions. The terminal would then perform all independent edit functions and be responsible for updating the screen. Up to this point no communication would occur between the terminal and MXE. When a key is pressed which requires 2200 interaction the terminal would inform the MXE and in the cases of Recall and Return would pass the entire edit buffer to the MXE. The terminal would then do nothing until the MXE responded. All keys pressed during this time would be stored in a terminal keyboard buffer but no editing would be done until the MXE commanded the terminal on its next course of action. Meanwhile, the MXE would confer with the 2200 on what to do. In the case of Recall the 2200 has the option to change the contents of the edit buffer after which it will allow the line request to pick up where it left off. The MXE would pass the new contents of the edit buffer to the terminal (if the buffer was changed by the 2200), position the cursor and tell the terminal to continue. All keystrokes in the keyboard buffer at that time would be acted on at that time.

In the case of Return the terminal would pass the MXE the contents of the edit buffer. The MXE would then pass that on to the 2200. The terminal does not exit the editor until the MXE tells it to do so. Thus the Remote Editor would be completely controlled by the MXE and 2200 at all times.

An extension to the above procedure could be made for use when traffic over the communications connection need not be minimized. In this case the terminal would pass all keystrokes to the MXE as they are entered thereby minimizing the delay time incurred when Recall and Return keys are pressed. The MXE would always have a copy of the edit buffer so the need for the terminal to pass its edit buffer to the MXE would be eradicated. This alternative procedure would use the same protocol as above.

Enabling the Remote Editor:

Each time the terminal sends a reset to the MXE it would follow it by a command byte which tells the MXE that the terminal supports some version of the extended protocol. This byte would not cause any problems when the terminal is connected to an MXD except when the VP operating system is used. This byte would then be turned into a text atom the 2200. This problem can be rectified by a changed to the 2200 VP operating system. This would be the last change to the VP OS of this type (a similar change was necessitated by the implementation of Remote Screen Dump) because the MXE would respond to this code by asking the terminal what it supports rather than defining a new code each time a new function is added. All terminals generating this code must support some minimum set of extended commands so the MXE could be sure it was legal to talk to the terminal in this way.

Several options are at this point possible. If the terminal does not have the RAM code space to be downloadable the terminal must have the Remote Editor in Prom. If the terminal is downloadable then the MXE and 2200 will be responsible for downloading code to the terminal. Currently, there is no way for the MXE to request data from the 2200 for downloading but an extension to the 2200-MXE protocol could be set up to allow this. Alternatively, if the set of downloadable terminals is kept small and the code needed for all these terminals is within some reasonable limit (which is a function of extra RAM space in the MXE) the download code for the terminal could be passed to the MXE at system configuration time and all terminal downloading would be transparent to the 2200.

Once the terminal is ready to act as a Remote Editor (ie it has been downloaded if necessary and the terminal has been command to use the extended protocol) all editing can be done using the terminal as editor.

The selection of editor mode could be automatic or manual. If manual, an MXE command mode command could be implemented to select the mode of remote editing desired (two modes: the MXE always keeps a copy of the screen or the MXE gets updated as needed). It is also be possible that criteria could be developed for the MXE to decide which mode it to be enabled. The criteria for auto mode selection could be changed as with future MXE code releases when it became necessary to respond to changes in communications technology.

Protocol Extensions:

This section contains a description of the added command structure at the protocol level.

The control byte from the terminal to the MXE after each reset would be a hex(E3). This code would signify to the MXE that the terminal supports some subset of the extended protocol. This byte is NOT escaped. The VP Operating System will interpret this byte as a $CLOSE text atom (using an MXD. The MXE will not pass this byte to the 2200).

The extended protocol adds new commands to the currently existing command structure from the MXE to the terminal. In this direction the command escape code is hex(FB) and is followed by one byte designating the command desired. No command structure in the opposite direction (from the terminal to the MXE) currently exists. A new command escape code of hex(FF) would be used in this direction.

Commands to the Terminal

Code	Meaning
FB C0	Select protocol mode
FB C1	Pass identification string
FB C2	Start download block
FB C3	End of block (any type of block
FB C4	Execute code
FB C5	Set up edit parameters
FB C6	Start block of edit data
FB C7	Begin edit
FB C8	Cancel edit
FB C9	Continue edit
FB CA	Add to edit buffer at end
FB CB	Start block of keystrokes for edit
FB CC	Pass edit buffer to MXE
FB D3	Start Remote Screen Dump Data *
FB D4	Stop Remote Screen Dump Data *
FB D8	Stop *
FB DA	Go *

NOTE: to send a data byte of hex(FB) the MXE sends a hex(FB DO)

Commands to the MXE

Code	Meaning
FF C0	Start device parameter block
FF C1	Start block or edit data
FF C2	Edit termination
FF C3	End of block (any type of block)
FF C4	

NOTE: to send a data byte of hex(FF) the terminal sends a hex(FF FF)

* These commands were added when Remote Screen Dump was added to the 2336DW terminal and would be considered part of the extended protocol.

Command descriptions:

Commands to the terminal

FB C0 xx xx ... FB C3 - Select Protocol mode

This command would be used to inform the terminal that the MXE will allow it to use the extended protocol. Parameters would also be passed to configure the terminal for the selected editing mode, etc, etc. This command can easily be extended by adding more parameter bytes as new functions are implemented. The string MUST be ended by an end of block command.

FB C1 - Pass Identification string to MXE

This command instructs the terminal to pass its identification string to the MXE thereby making the MXE aware of the terminals capabilities and limitations.

FB C2 - Start Download Block

This is the start of a block code to the terminal. Each block of code consists of a START command followed by a TWO or FOUR BYTE ADDRESS (depending on the terminal) followed a variable number of data bytes followed by an END OF BLOCK command (FB C3).

FB C3 - End of Block

This command signals the end of a block of any type.

FB C4 xxxxxxxx - Execute Code at Location xxxx

"xxxxxxxx" is the four byte address of the start of the downloaded code. On terminals with only two bytes of address space the first two bytes will by zero.

FB C5 xx xx ... FB C3 - Set Up Edit Parameters

This command designates to the terminal all parameters needed to execute the next edit. The string of parameters must be terminated by the end of block command FB F3. The terminal should stop sending keystrokes to the MXE at this point. All keystrokes should be saved in a keyboard buffer for later use when the edit begins.

FB C6 - Start Block of Edit Data

The MXE uses this command to start the data block which is to be used as the contents of the edit buffer. This command may be used anytime the MXE sees fit. The previous contents of the edit buffer is completely wiped clean but the screen is not changed until the MXE sends a continue edit or begin edit command.

FB C7 - Begin Edit

This command starts the editing process. Until this command is issued the terminal should not have manipulated the screen in any way.

FB C8 - Cancel Edit

The current edit is cancel and the screen is updated. All bytes are passed to the MXE as they are typed until the next edit. The contents of the edit buffer are should not be changed until the next command from the MXE which operates on the buffer.

FB C9 - Continue Edit

This command causes the editing procedure to continue from exactly where it left off. This command is generally used after a special function or recall. The screen should be updated.

FB CA xx xx ... FB C3 - Add to the Edit Buffer

This command is the start of a string which is to be added to the end of the edit buffer. The end of block command FB C3 must be used to terminate the string. The screen is not updated until the MXE sends the continue edit command.

FB CB xx xx ... FB C3 - Start Block of Keystrokes for Edit

The string started by FB CB and ended by FB C3 is to be used as keystrokes from the keyboard would be. All bytes must be valid codes which can be generated from the keyboard. This string is usually the contents of the MXE keyboard buffer when the edit command was received from the 2200. The string should be placed in the keyboard buffer in front of all bytes in the buffer at the time.

FB CC - Pass Edit Buffer to the MXE

This command instructs the terminal to pass the current contents of the edit buffer to the MXE. The terminal should respond with command FF C1.

FF C0 xx xx ... FF C3 - Start Device Parameter Block

This command signals the start of the device parameter string. The string must be terminated by the end of block command FF C3. The string contains information about the device capabilities and limitations (such as remote edit support, remote screen dump support, etc.). Since the block is variable length new functions can easily be added in the future.

FF C1 xx xx ... FF C3 - Start Block of Edit Data

This command signals the begining of an exact copy of the edit buffer in the terminal. This block must be terminated by end of block command FF C3. The MXE will replace the contents of its edit buffer with this block. The block can be no longer than 480 bytes.

FF C2 xx - Edit Termination

This command signals the termination of the edit in the terminal. The byte immediately following the command describes the type of termination. If the termination was caused by a return or recall then the terminal will follow this command by sending the contents of the edit buffer using command FF C1. Otherwise the terminal will take no further action until the MXE instructs it to. All editing will cease until the MXE instructs the terminal of its next action.

FF C3 - End of Block

This command signals the end of a block of any type.

Reset and Halt/Step

Reset (hex(12)) and Halt/Step (hex(13)) in the edit data must be escaped with transparency escape hex(FC) or they will be interpreted as other than data. This is in keeping with the conventions already established in the existing protocol.

DOCUMENT SUMMARY

Document Id: 0081E
Document Name: Trip Report, Germany 6/83
Operator: Eric
Author: Eric

Comments:

Pages to be printed 5

Notify U13 on system PAM.

Subject: Trip report - Trip to Germany to implement the X.25 network temporary solution
From : Eric Wilson
Date : 30 JUNE 1983
Distribution:

Neeraj Sen	MS 1489
Bruce Patterson	MS 1389A
Jerry Sevigney	MS 1389A
Rich Racicot	MS 1489
John Deutsch	MS 14A7A
Peter Thornton	MS 13A3B
Tim Sloane	MS 13A3B
Heinz Jurack	Wang Deutschland

This report is the result of a trip taken by the author in June of 1983 to Germany for the purpose of implementing a solution to the "Terminal Echo Problem" over an X.25 network. All comments should be sent to Eric Wilson at MS 1483, Tower ext 7192.

Purpose of the trip

Using a 2200 terminal (2236, 2336 series) over an X.25 network proved to be inefficient and expensive due to the system packet transfer delay and the volume of packets used respectively. Normal editing functions were being performed over the network thus requiring a long period of time for a character to be echoed to the screen and at least two packets to be transferred. This was unacceptable to Shell Oil of Germany and British Petroleum of Germany so the "Echo Off" solution was devised to ease the situation. This solution was devised as and is still considered a SHORT TERM solution until WANG is able to provide a long term COMPLETE solution at a later date.

For a full description of the proposed solution please see "Short term TEMPORARY solution to the german X.25 packet switching network problem" 13 MAY 1983 by Eric Wilson.

Solution Shortfalls (as implemented)

The solution implemented in Germany was based upon the above mentioned document with some modifications due to MPAC limitations and timing problems encountered during testing. Also, the severity of some limitations stated in the solution proposal was lessened through MXE functionality added during testing. One timing problem was found to be unsolvable although not critical. All solution limitations and enhancements are listed below along with their status.

1) It is possible to type outside the proper edit field causing the screen image of the edit field to be different then the MXE image (the MXE image will be sent to the 2200 at the end of the edit). This happens because the MPAC has no knowledge of edit fields and simply echos all bytes as they are typed. To help rectify this problem the INSERT, DELETE, BEGIN and END keys perform the added function of reprinting the contents of the edit field before performing their usual function. In this way the user can always find out exactly what will be sent to the 2200 at any time.

2) Some keystrokes require action by the MXE or the 2200. As this response must be transmitted over the X.25 network it is possible for a user to type ahead of the response thereby causing events to occur in the wrong order on the screen. To prevent this from happening the MPAC waits for a response from the MXE after sending keystrokes which it knows may require a response. All responses from the MXE have a HEX(0000) appended to the end to signal to the MPAC the end of the response. Until the HEX(0000) is received by the MPAC no local screen editing is performed and keystrokes are buffered. The MPAC keeps a count of the number of outstanding responses so that local editing does not resume until all responses are received.
The utility programmer should be careful not to use HEX(0000) within LINPUT prefills so as not to confuse the MPAC (although this is a highly unlikely case as HEX(0000) serves no purpose in normal LINPUT data streams).

3) An unsolvable (NON-CRITICAL) MPAC timing problem exists as follows: If the user were to hold down either the left or right cursor key on a DW terminal the MPAC would continuously respond by echoing a move cursor command to the terminal. The cursor keystrokes would be buffered in the MPAC until a full packet were formed thereby necessitating transmission to the MXE. When the MPAC sent the packet to the MXE it would loose one byte from the terminal (because the bytes are coming to fast) thereby missing one half of a cursor keystroke from the terminal (cursor keystrokes are two byte escape codes). The MPAC would then see the next byte (or the byte following the next byte) as a one byte code and would echo it to the screen instead of a move cursor command. Thus, an extraneous character would appear in the edit field. Of course, there is no reason for the user to hold down the cursor key for so long a time. Although this problem does not always occur when the unshifted cursor key is held down it will almost always occur when a shifted cursor key (5 left or right) is held down. In this case the cursor must wraparound on the screen at least 3 times!

4) It was previously thought that only DW type keyboards are used on the system remotely. If this were the case than the function keys need never be used for cursor movement. As this is NOT the case a modification needed to be made so that the MXE would respond over the network to update the screen. This function was not added in Germany due to equipment malfunctions on the last day before my return to Lowell. However, this fix will be sent to Germany shortly (Heinz Jurack is currently visiting WANG Lowell and will take the new version back with him).

Code Testing

As a final test before my return to Lowell the MXE and MPAC code was installed at a Shell computing center near Munich. The system used was an MVP with one MXE and two MXD controllers connected to four remote terminals, two local terminals and a 300 baud modem used for diagnostic purposes by Shell. One remote terminal was operated using the new code configuration and two local terminals as before. One of the local terminals was connected to the MXE and the other to an MXD.

Using Shell's standard network software the new configuration was found to use packets far more efficiently and response time to most keystrokes was the same as a local terminal. The editing functions INSERT, DELETE, BEGIN and END still required MXE action so these functions performed slower (ie with a network delay).

It was at this stage of the testing that a DW keyboard was connected remotely and the lack of cursor control noted.

It was not possible to accurately measure such system parameters as packet usage and network delay due a lack of measuring equipment. Estimates can be made theoretically but exact figures will vary according to the application being used. No such conjectures will be made in this report.

Code Status

Shell Germany will test the new code at a selected computing center for a two to four week period before installing it at all their german sites. Testing will have already begun by the writing of this report. WANG Germany will control the MXE code within germany and will absolutly not release the code outside germany without the written consent of WANG Lowell R&D.

No written legal agreement has been made with Shell Germany or TDT thus far.

Time Usage

No log was kept of WANG resources utilized on the project. Some figures are known and are presented here:

Time in germany:	2 weeks = 2*5days/week	10 days
Prep time in Lowell:	3 days	3 days
Wrap-up time in Lowell:	4 days	4 days
Total days		17 days
Total hours	17days * 8hours/day	136 hours

This is an approximation and does not take into account much time spent in various related activities, other WANG personnel nor WANG equipment usage.

Participants

Wang Lowell:

Eric Wilson	Software Engineer II
Tim Sloane	Market planning

Wang Deutschland GMBH:

Heinz Jurack	Manager, Distributed-Information-Processing (DIP) Telecommunications and Networking

Shell Oil Company of Germany:

Herr Griem	Computer systems
Frau Holz	Systems analyst

TDT (Transfer Data Test GMBH):

Antherm Pickhardt	Geschäfstführer (Manager)
Gerhard Riedl	DFÜ-techn.-Unterstützung (Technical support technician)
Herr Bartel	DFÜ-techn.-Unterstützung (Technical support technician)

DOCUMENT SUMMARY

Document Id: 0085E
Document Name: TC extension proposal
Operator: Eric
Author: Eric

Comments:

Pages to be printed 8

Notify U13 on system PAM.

From: Eric Wilson
Subject: MXE TC extension proposal
Date: 25 July 1983
Distribution:

Rich Racicot	
Bruce Patterson	
Jerry Sevigney	
John Deutsch	MS 14A7A
Gene Mantoni	MS 14A7A
Tyler Olsen	MS 1459

All comments on this proposal should be directed to Eric Wilson @Tower x7192, MS 1489.

This proposal is written in response to several user requests for extensions to the MXE TC package. It is intended as an extension to the already existing TC package and would require minimal code changes.

No flow control options are currently supported by MXE TC. The only flow control currently supported is the interpretation of CTS as output data throttle and DCD as input data validator. These interpretations are automatically assumed by the MXE upon configuration. This proposal would allow the user to select from two forms of flow control in each direction including Xon/Xoff and various control signals.

It is currently not possible to detect the state of many of the RS 232 signals. This proposal would allow the current state of DCD, RI, DSR, CTS, DTR and RTS to be read by the user.

It is often necessary to use some RS 232 signals in non-standard ways. This proposal allows direct user control over RTS and DTR.

The attached appendix contains a description of the current Status Vector, Control Vector and command formats. The following extensions apply to that description.

Status Vector extension

An eighth byte will be added to the status vector which contain the current state of all control signals supported by the MXE (in both directions) as follows:

bit	pin	meaning
01	8	DCD (Received Line Signal Detector)
02		Always zero
04	6	DSR (Data Set Ready)
08	5	CTS (Clear To Send)
10	20	DTR (Data Terminal Ready)
20	4	RTS (Request To Send)
40	22	RI (Ring Indicator)
80		Always zero

Please note that the bit positions for each signal in status byte 2 map directly on to the above byte.

Also: the user should be careful in his/her interpretation of this new status byte with respect to bits 02 and 80. If future extensions were to be added to this byte these bits could take on new meanings causing them to change from zero. Bits 02 and 80 are guaranteed to be zero for this extension.

The user may read in only 7 bytes if desired so that all old applications will not be affected by this extension.

Communications Control Vector extensions

Seven bytes will be added to the communications control vector. These bytes will be appended to the end of the currently existing control vector with no changes to current byte definitions. The seven bytes are as follows:

Byte	Bit	Meaning
21**	01*	select CTS/DCD as flow control from connected device
	02*	select DTR as flow control from MXE
	04***	select Xon/Xoff as flow control from connect device
	08***	select Xon/Xoff as flow control from MXE
	10	select direct control of DTR from basic program
	20	select direct control of RTS from basic program
	40	always zero
	80	always zero
22		flow control escape from connected device for Xon/Xoff two byte escape sequences. If this byte is zero then a one byte flow control code is assumed
23		Xon code from connected device. *** If byte 22 is NOT zero then this code must be preceded by escape byte 22 to be interpreted as an Xon command.
24		Xoff code from connected device. *** If byte 22 is NOT zero then this code must be preceded by escape byte 22 to be interpreted as an Xoff command.
25		flow control escape from MXE for Xon/Xoff two byte escape sequences. If this byte is zero then a one byte flow control code is assumed
26		Xon code from MXE. *** If byte 25 is NOT zero then this code must be preceded by escape byte 25 to be interpreted as an Xon command.
27		Xoff code from MXE. *** If byte 25 is NOT zero then this code must be preceded by escape byte 25 to be interpreted as an Xoff command.

If the user does not wish to use any of the above functions then only the first 20 bytes of the control vector need be loaded. All old applications will therefor not be affected by this extension.

If Xon/Xoff is not being selected but flow control via DTR is desired the user need load only byte 21 with the appropriate bits set.

If Xon/Xoff is desired the user should load ALL bytes up to byte 27.

If the user selects Xon/Xoff from the connected device and uses a two byte escape sequence only valid Xon/Xoff codes will be deleted from the data stream. Two byte sequences which are not valid flow controls will be passed to the users basic program.

When using Xon/Xoff flow control the MXE will send Xon/Xoff controls even when the connected device has sent an Xoff code.

When using DTR flow control the MXE will accept data from the connected device even when DTR is inactive. It should be noted however that buffer overrun may occur if data is received when DTR is low.

The MXE will NOT send Xon/Xoff if CTS is low and CTS/DCD are selected as flow controls.

Xon/Xoff codes will not be recognized when CTS/DCD is selected as flow control and DCD is low.

* The MXE normally interprets CTS as output flow control and DCD as input data validator signal.

** Caution should be used when designating options in byte 21 (especially multiple flow control in one direction or mixed combinations)..

*** If Xon/Xoff is selected and any of bytes 23,24,26,27 are zero then the MXE will assume no flow control in the direction where the Xon/Xoff codes are not completely defined.

Command Extension

A new command to set the states of DTR and RTS is to be added as follows:

CBS(10) OBS(xxyyzz) CBS(0C)

Where: bit 10 of xx sets the state of DTR
bit 20 of xx sets the state of RTS
byte yy designates which bits in xx are to be ignored
byte zz designates which states set in xx are to be locked

Byte xx designates the states which DTR and RTS should be set to. 1 is the active state (positive voltage). Byte yy is a mask which designates which bits in xx are valid. It is therefore possible to set one or both signals at the same time. Byte zz allows the option to lock these states so that no other condition in the MXE may change them (including flow control and opening output channels). Bits 10 and 20 of yy and zz correspond to bits 10 and 20 of xx.

This command will set DTR and RTS the the states desired no matter what flow control option is currently selected. However, if DTR is optioned to flow control this command will not set the DTR bit permanently unless the lock bit in zz is set. When the MXE decides to set DTR to a specific state due to flow conditions it will do so unless byte zz has specified that the state set is a permanent one in which case the MXE will not change DTR until told to do so.

The disconnect command will work as it does currently by setting DTR inactive for 3 to 5 seconds and then setting it to active. The DTR lock will be cleared by the disconnect command.

RTS is set active each time the user opens the output channel. RTS is set inactive automatically when a line turnaround occurs in half duplex. If byte zz has locked the state of RTS then RTS will not be changed by the MXE until told to do so.

The user need not output all three bytes to the MXE each time a signal is changed. All three bytes (xxyyzz) are saved by the MXE and are overwritten by each time the command is issued. The user may overwrite 0, 1, 2 or all 3 bytes as desired. The signal states on the RS 232 connector are not changed until the CBS(0C) is received by the MXE. If the CBS(0C) is not sent then the current signal states are not changed. It is therefor possible to lock the current state of DTR and RTS by writing 3 bytes to the MXE with the third byte containing the desired lock information but writing a CBS(00) instead of a CBS(0C) to end the command without setting new signal states (beginning a new command after the third byte will produce the same results but the user should be careful to reopen the output channel before resuming output).

2227B TC board compatibility

These new functions are not being proposed for the 2227B communications controller. All programs which run using the MXE without these extensions will run on the 2227B. Any attempt to use these functions on a 2227B will be IGNORED by the 2227B as any command outside the currently defined range is ignored as are command bytes past 18 in the command vector.

Manhours

Two manweeks will be required to code and test the proposed changes (not including Q&A).

APPENDIX