Application Note

UniSystem Computer Remote Control

This Application Note describes how to use computer remote control (CRC) commands to remotely control your UniSystem programmer (UniSite-*xpi*/UniSite, 2900, 3980*xpi*/3980/3900, AutoSite, or ProMaster 2500).

Note: This Application Note applies to UniSystem programmers running system software ***Version 6.8*** *or higher.*

The most likely reason for controlling your programmer in remote mode is that it allows you to use a custom user interface.

This Application Note is divided into the following sections, which appear on the following pages:

Note: This Application Note is intended to help those Data I/O customers who are using or developing a custom driver program to control a UniSystem programmer in remote mode.

If you are using your programmer in terminal mode or if you are using PROMlink™ or TaskLink™ Automation Software to control your programmer in remote mode, you do **not** *need to read this Application Note.*

For information about PROMlink or TaskLink, refer to the PROMlink User Manual or TaskLink User Manual. *For information about terminal mode, remote mode, and UniSystem programmers, refer to your programmer's User Manual.*

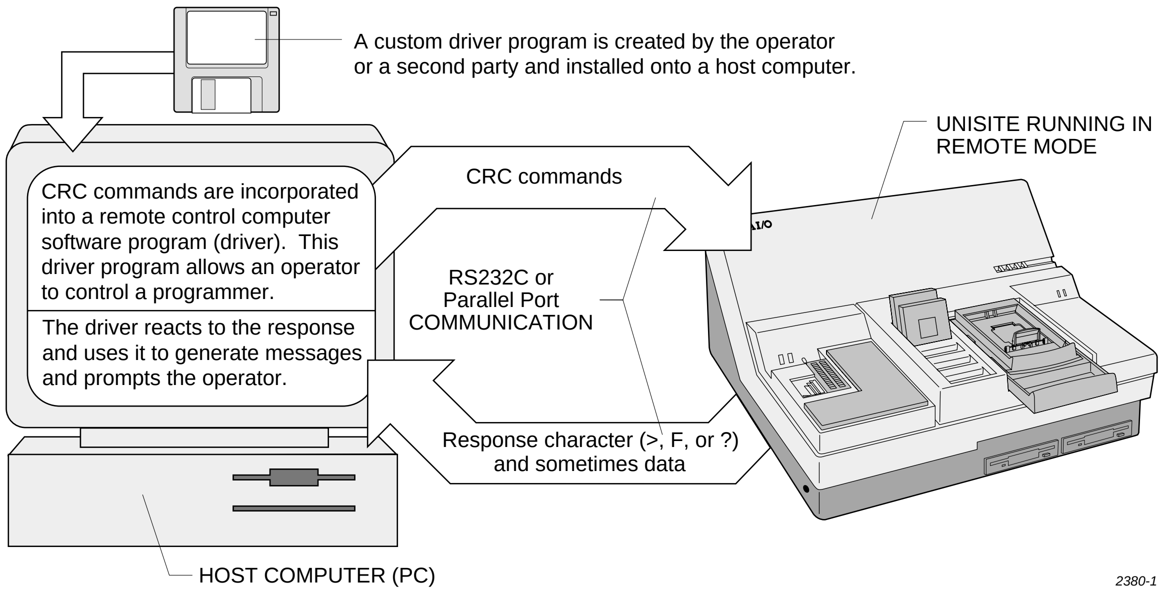
983-0490-005

Introduction

How Do CRC Commands Work?

Figure 1 summarizes how CRC commands work.

Figure 1. Interaction between a Custom Driver Program and UniSystem Programmer (UniSite shown).


A custom driver program is created by the operator or a second party and installed onto a host computer.
UNISITE RUNNING IN REMOTE MODE
CRC commands
CRC commands are incorporated into a remote control computer software program (driver). This driver program allows an operator to control a programmer.
The driver reacts to the response and uses it to generate messages and prompts the operator.
RS232C or Parallel Port COMMUNICATION
Response character (>, F, or ?) and sometimes data
HOST COMPUTER (PC)
2380-1

As shown in Figure 1, CRC commands are incorporated into a remote control computer software program (**driver**), which is installed on a host computer. The driver sends CRC commands to the programmer, which is running in remote mode. This driver is created by you or a second party using the information presented in this Application Note.

There are two types of drivers:

- **Simple drivers** require the use of a terminal emulator program to communicate with the programmer. With this type of driver, you run your terminal emulation software and, at the prompt, you can type in the CRC commands or you can send a text file that contains CRC commands.
- **Complex drivers** include the terminal emulation (or, for *xpi* programmers, an optional parallel interface) software within the code. One example of a complex driver is TaskLink software, which includes a full menu-driven user interface as well as the ability to communicate with the programmer.

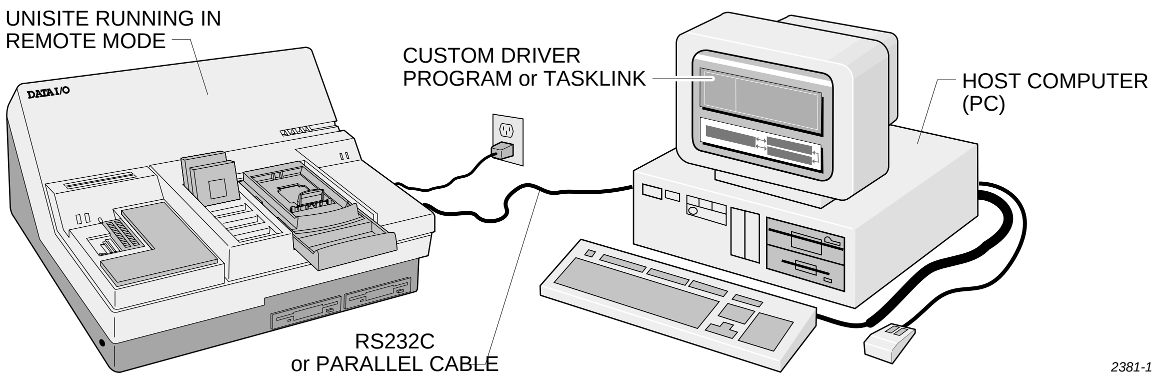
For more information on drivers, see "Creating A Custom Driver Program" on page 7.

Getting Started

System Setup

Before you can use CRC commands, your programmer system must be set up in remote mode (see Figure 2).

Figure 2. Typical Configuration for a PC Sending CRC Commands to a Programmer (UniSite)


UNISITE RUNNING IN REMOTE MODE
DATA I/O
CUSTOM DRIVER PROGRAM or TASKLINK
HOST COMPUTER (PC)
RS232C or PARALLEL CABLE
2381-1

To set up your programmer system in remote mode, do the following:

1. Turn off the power to your PC and programmer.
2. Set up your programmer and PC as described in the "Setup and Installation" section of your **Programmer's User Manual**. This includes connecting an RS232C cable (for serial communication) or parallel cable (for optional high speed communication with *xpi* programmers) between your PC and programmer. The serial cable must be connected to the **Remote** (**Handler**) port on the programmer.

   Note: In this Application Note, references to the **Remote** *port apply to the* **Handler** *port and references to the* **Terminal** *port apply to the* **Auxiliary** *port on AutoSite, AutoSite/E, and 2500.*

3. Turn on the power to your programmer and PC.
4. If your programmer is not already set up to run in remote mode, after the programmer boots and completes its powerup self-test, set the programmer to operate in remote mode as described in the "CRC" section in your **Programmer's User Manual**. (These steps are summarized in "Entering Remote (CRC) Mode" below.)

   Note: To ensure correct operation of the remote port with the host computer, set the parameters for the remote (Handler) port according to the host computer requirements.

Entering Remote (CRC) Mode

If your programmer is booted and running in terminal mode, to enter remote mode follow these steps:

1. Press **F1** to get to the Main Menu.
2. Press **M** to select the More Commands menu.
3. Press **R** to select Computer Remote Control from the More Commands menu.

The programmer is now in remote control mode.

Note: In remote mode, the programmer will accept CRC commands from the host computer connected to the programmer's ***Remote*** *port or* ***Parallel*** *port (for xpi programmers).*

Entering Remote Mode on Powerup

If you wish to set up your programmer to enter remote mode during powerup, do the following:

Note: Your host computer (which generates the CRC commands) must be connected to either the Remote (Handler) Port or the Parallel Port (for xpi programmers) on your programmer. CRC will not work on the Terminal (Auxiliary) port. If your host computer is connected to the programmer's Remote port and no terminal is attached to the programmer's Terminal port, the programmer enters remote mode automatically at powerup. If the Parallel port is the only port connected, this will also cause the programmer to enter remote mode automatically on power up of an xpi programmer.

1. After your programmer has finished booting in terminal mode, press **F1** to get to the Main Menu.
2. Press **M** to select the More Commands menu.
3. Press **C** to select the Configure System menu.
4. Press **E** to select Edit from the Configure Systems menu.
5. Press **I** to select Interface from the Edit menu. The programmer displays the interface parameters.
6. Move the cursor to the **Power on CRC field** and press **Y**.
7. Press **F2** two times to return to the Configure System Parameters menu.
8. Press **S** to select Save from the Configure System Parameters menu. The screen displays the Save System Parameters menu.
9. Press **1 ENTER (RETURN)** to select the Powerup Defaults file as the one in which system parameters will be saved.
10. Press **ENTER (RETURN)** again so that the selection will be saved to the disk.

The next time you power up the programmer, it will enter remote mode automatically.

Exiting Remote Mode

To exit remote mode, send the **Z** (**Z ENTER**) command to the programmer. When you exit remote mode using the **Z** command, the programmer's parameters are reset to the way they were BEFORE you entered remote mode.

---

*Note: If you have a terminal attached to the **Terminal** port, and the User Port is set to **T** (for Terminal/Auxiliary port), you can also exit remote mode by pressing* **CTRL + Z** *on the terminal attached to the Terminal port.*

When you exit remote mode using **CTRL + Z** *on the terminal attached to the Terminal port, the programmer's parameters remain the same as they were in remote mode.*

Suspending Remote Mode

Remote mode can be suspended temporarily to allow you to go into terminal mode to view or change parameter settings or the device data in memory. To suspend remote mode, send the **49]** command. See the description of **49]** on page 35 for more information.

Halting CRC Operations

To halt any command or any ongoing CRC operation, use one of the following commands from the remote port or the parallel port. Neither of the following two commands requires a **RETURN** (**ENTER**). Both commands are immediate and both terminate any preceding command operation.

ASCII Command	**Hex Code**	**Description**
ESC	1B	Causes the programmer to unconditionally halt any operation except a binary transfer.
BREAK	N/A	(Remote port only). Causes the programmer to unconditionally halt any operation in progress. This includes all data communications transfers. The data line must be held in the spacing condition for 110 ms to 700 ms.
DIRECTION	N/A	(Parallel port only). Changing the state of the DIRECTION line on the parallel port causes the programmer to unconditionally halt any data communications transfers. See "XPI Parallel Port Interface" on page 57.

Using Remote ON/OFF to Connect Multiple Programmers

You can use Remote ON/OFF to connect multiple programmers to a single host computer by doing the following:

1. Connect one programmer to your host computer.
2. Boot the programmer in terminal mode and press **F1** to get to the Main Menu.
3. Set the **Remote On Code** and **Remote Off Code** for the programmer in the **More Commands/ Configure System/Edit/Interface Parameters** menu. The Remote On/Off codes are usually set to 0, which means that they are disabled. To enable Remote On/Off codes, set them to any ASCII character other than zero (0). When selecting the Remote On/Off Codes for your programmer, make sure that the code is not the same as data that might be sent to the programmer.

Note: *The Remote On/Off Codes are usually set to control characters (hex 01 to 1F) and are used when transfers contain non-binary data.*

When the Remote On/Off Codes are set to something other than zero, the programmer ignores all data input after receiving the Remote Off Code until it receives the Remote On Code. After receiving the Remote On Code, the programmer processes commands and data normally until another Remote Off Code is received.

4. Set up your programmer to enter remote mode as described on page 4.
5. Disconnect your programmer and repeat steps 1 through 4 for each programmer.

Note: *To avoid possible data transmission conflicts, we recommend that you give each programmer its own unique Remote On/Off Codes.*

6. Connect all the programmers to your host computer.

CRC Default and User-Defined Settings

When remote mode is entered, certain defaults are set prior to the programmer's accepting any commands. The default settings are outlined below:

Description	Setting
Upload/download port	Remote port
Data source/destination	RAM
Security fuse data (0 or 1)	0
Program security fuse	No
Reject option (commercial or single)	Commercial
Algorithm Type	D (Standard Algorithms)
Logic verification option	All
Number of verify passes (0,1, or 2)	2
Fill RAM before downloading	No
Illegal bit check option	No
Blank check option	No
Enable yield tally option	No
EE bulk erase option	No
Odd/even byte swap for 16 bit option	No
JEDEC I/O translate DIP/LCC option	Yes
Continuity check option	Yes
Compare electronic signature	Yes
Host command	Blank
I/O address offset	FFFFFF
I/O format	MOS technology (format 81)
Instrument control code (0,1, 2)	0
I/O timeout	30 seconds
Upload wait	0 seconds
Number of nulls	255
Serial set auto-increment mode	No
Programming mode	Single device
Total set size	1
Upload EOF delimiter flag	Disabled
Download EOF delimiter flag	Disabled

When you exit remote mode using the **Z** command (sending **Z ENTER**), the programmer's parameters are reset to the way they were BEFORE you entered remote mode. If you wish to keep your CRC parameter settings, you need to first use the **FE]** (save user-defined CRC parameters) command and, each time you enter remote mode, use the **FD]** (restore user-defined CRC parameters) command.

If you exit remote mode by pressing **CTRL** + **Z** (from a terminal attached to the Terminal port), the programmer's parameters remain the same as they were in remote mode.

Creating A Custom Driver Program

The following sections describe how to create a custom driver program:

- **Getting Started with CRC —** Describes how to download data from your PC to the programmer using CRC commands through an RS232C connection.
- **Example —** Shows an example of CRC commands used to download data as described in the "Getting Started" section.
- **Response Characters —** Describes what the driver can expect as a response from the programmer.
- **Exiting Code —** Describes how your driver can tell the programmer to exit from remote mode.

Getting Started with CRC

The remote mode (CRC) command/response protocol is roughly the same whether you use the serial remote port or the *xpi*'s parallel port. This section will concentrate on the serial port. Terminal emulation software is readily available for the serial port. However, there is no analog for the parallel port. Using a terminal emulator with remote mode on the serial port is a good way to become familiar with the CRC command/response protocol before writing your own driver.

After you connect the RS232C cable to your programmer as described in your **Programmer's User Manual**, the host computer needs to be running **terminal emulation software** capable of emulating a **DEC VT-100** in order to communicate with the programmer.

Terminal Emulation

For RS232 serial communication, choose a terminal emulation program such as the **HyperTerminal** program included with Microsoft Windows, or create your own terminal emulator program. Set the program to emulate a **DEC VT-100**. Initialize the host computer's port that is connected to the programmer (usually COM1:) so that its settings are the same as the port settings on your programmer. In most cases, set the serial port on your host computer to:

9600 Baud	8 bit data
No parity (parity off)	Xon/Xoff handshaking
1 stop bit	

Note: *If you have any problems, make sure the programmer is in remote mode (and not disabled using the Remote On/Off Codes), make sure the port settings are the same for the terminal emulator and programmer, and make sure the cables are properly connected.*

You might find it useful to set your terminal emulator to ***add LFs*** *to the data sent to and from the programmer (outbound and inbound data). If you are sending text files containing CRC commands to your programmer, you might also find it useful to set your terminal emulator to send* ***one line at a time*** *and to* ***strip line feeds*** *(LFs) from the file.*

Establishing First Contact

When first establishing contact with the programmer, it is usually a good idea to send a Display Configuration command (**01]**). This command will return a string of characters as follows:

RRR/SSSS/AAAA/MM/PPP/IIVV/JJVV/KKVV/QQ> **OR** *SSSS/AAAA/MM/PPP/KK/QQ>*

where *RRR* is the current ROM version (UniSite only), *SSSS* is the current system version, *AAAA* is the Algorithm version, *MM* is the decimal number of 64K-byte banks of RAM memory available, *PPP* is the decimal number of pin drivers available, *II* identifies the installed PSM (UniSite only; *VV* is the version number), *JJ* identifies the installed FSM (UniSite only; *VV* is the version number), *KK* is a hexadecimal number identifying the installed Base (*VV* shows the version number on UniSite only), *QQ* is the decimal number of the PPI adapter installed (if *QQ* is 00, no PPI adapter is installed).

Note: The string returned by the view configuration command will be terminated with the response character ">" and a carriage return.

A typical session might be to download a formatted data file output by a linker/locator to the programmer, and then to program a device. A typical sequence follows:

1. **Select the device first.** This is always a good idea, because selecting the device sets default values and gives the programmer some basic information.
2. **Set user data size, I/O offsets, or beginning RAM address** if any, as these affect downloading to the programmer.
    - ♦ **User Data Size** is a hexadecimal value (usually set to the device size in bytes or to a multiple of the device size in bytes for set operations) that is used to determine the data block used in device operations. The data block is the range in user RAM where you have specified data can be stored. Any data outside the data block is truncated (not stored). The data block is defined by the Beginning RAM Address (start of the data block) and the User Data Size, which is added to the Beginning RAM Address to determine the end of the data block. For example, if Beginning RAM Address was set to 006000 and User Data Size was set to 004000, the data block where data is stored would be from 004000 to 009FFF (any data record sent to an address outside this range is discarded).
    - ♦ **I/O Offset** specifies how much of an offset is used during a data transfer. The I/O offset is added to all addresses sent out from the programmer (data uploaded or sent to a disk drive). The I/O offset is subtracted from all addresses received by the programmer (data downloaded or from a disk drive). For example, if you set I/O offset to 001000, the data downloaded to the programmer would be placed at the address specified by the data record minus 001000 in user RAM.

      If you set I/O offset to 000000, all data is stored at the address specified by the data (since 000000 or zero is subtracted from the address of the incoming data record).

      If you set I/O offset to the default (FFFFFF), the programmer sets the I/O offset to the first incoming address of data received from Input from Disk and Download operations. For example, if I/O offset is set to the default of FFFFFF, and the first data record is downloaded with a specified address of 001000, the I/O offset is set to 001000.
    - ♦ **Beginning RAM Address** specifies the address at which to begin storing data. For example, if you specified a beginning RAM address of 001000, data with an address of 000000 (in the formatted data file) would be stored at RAM address 001000.
3. Select the I/O format of the file to be downloaded (this example uses Intel MCS-86 Hex format). For the programmer, this would be format 88.

   Note: Translation formats are described in your Programmer's User Manual.
4. Fill programmer RAM with 00 or FF so that you have a known starting point before downloading your data. If you know the checksum of your data file, you may want to fill RAM with 00 so that the checksum is computed only on the data loaded from the file.
5. Download the data file to the programmer.
6. Insert the devices in the programmer sockets.
7. Send a Program command to the programmer.
8. If the Program command failed (an "F" was returned by the programmer), inquire about the failure (see example on next page).

Example

The "Getting Started with CRC" sequence (described above) would look like this:

Host Sends	Programmer Returns	Comment
01]	*RRR*/*SSSS*/*AAAA*/*MM*/ *PPP*/*IIVV*/*JJVV*/*KKVV*/*QQ*> ***OR*** *SSSS/AAAA/MM/PP/KK/QQ*>	Establish contact. For example, on a UniSite programmer *RRR*/*SSSS*/*MM*/*PPP*/*IIVV*/ *JJVV*/*KKVV*/*QQ* could be 013/0480/0480/40/ 028/02/00/00
AMD33]	>	Select AMD devices.
27C51234]	>	Select 27c512.
000000<	>	Set Beginning RAM Address to 000000.
000000;	>	Set User Data Size and Device Block Size to the physical size of device (0x10000 bytes for the 27C512). Setting user data size to 000000 (zero) sets the user data size to the default size of the device.
000000:	>	Set beginning device address to 000000.
X	varies	View and clear any previous error status.
000000W	>	Set I/O offset to 000000.
88A	>	Select I/O format 88 (Intel MCS-86 Hex).
00^	>	Fill RAM with 00s before downloading data.
I		Send Input command, then start sending the data file to the programmer. Once the entire file has been downloaded to the programmer, the programmer will send back a ">" or "F".
S	*XXXX*>	Optionally send checksum command. Programmer will return the hexadecimal checksum value. *XXXX* is a hexadecimal number.
P	> or F	Send program command.

If the programmer returned a failure response ("F"), you will want to inquire about the failure. There are several different ways to get error information. For instance, you can do the following:

X	*XX*,*YY*,...*ZZ*>	Programmer returns hexadecimal error codes, then clears the errors. *XX*, *YY*, *ZZ* are hexadecimal numbers.
/ *(UniSite Only)*	*XXYY*>	Programmer returns the number of devices that failed, and the number of devices in the sockets. *XX* is the number of devices that failed. *YY* is the number of devices in the sockets.

Host Sends	**Programmer Returns**	**Comment**
DF] *(UniSite Only)*	*AA BB CC DD EE FF GG HH...>*	Programmer returns the status of all the sockets. The status is output as a string of hexadecimal numbers, the first being status for socket 1, the last for the last socket. The numbers indicate status as follows: Bit 7 = error Bit 6 = non-blank device error Bit 5 = device testing or overcurrent error Bit 4 = invalid electronic ID error Bit 3 = illegal bit error Bit 2 = programming error Bit 1 = verify error Bit 0 = device detected

Optionally, you can instruct the programmer to perform testing before the "P" Program command (so that you know all sockets contain programmable devices before issuing the Program command). To perform testing before programming, do the following:

DC]	> or F	Programmer performs a device check that includes testing for correctly inserted devices. Some devices are also tested for continuity or shorts.
B	> or F	Programmer performs a blank check of the devices. If all devices are blank, they should be programmable.
T	> or F	If the Blank Check command returned an F, you may send a T command to see if the devices can still be programmed. The illegal bit test checks to see if the non-blank devices can still be programmed.

After programming, you may optionally send a Verify command "V" to perform a separate verify.

Response Characters

When a CRC command is sent to the PROGRAMMER, it returns a response character to the host computer that indicates if the PROGRAMMER accepted or executed the command sent to it, failed during execution of the command, or did not understand the command. The response character is followed by a carriage-return. The response characters are:

>	Indicates command has been accepted and/or completed
F	Indicates the command has failed
?	Indicates the command is invalid or not understood

Table 1 explains each of these response characters. Depending on the command, the response can also include data, a line feed, and null characters (ASCII 00). If you want to ensure that a line feed is always sent after a CRC command is entered, set the null count between 00 and FEh (the h following the number designates a hexadecimal number and is not part of the number). The default null count is FFh, which does not send a line feed after the command is entered.

Table 1
Computer Remote Control Response Characters

ASCII	Hex	Description
>	3E	**Prompt —** A prompt is sent when you enter CRC after an ESC or BREAK has halted a command, or after a CRC command has been completed.
F	46	**Fail —** If the programmer responds with an **F**, the programmer has failed to run the last command entered because errors were generated. You can find out more about the errors by using one of the error inquiry commands, such as **X** or **/** (the / command is available on UniSite only).
		Note: The **X** *command causes the programmer to send back a complete list of the last sixteen error codes. The* **/** *command causes the programmer to respond with the total number of devices that failed, and the number of socketed devices.*
?	3F	**Question Mark —** If the programmer responds to the command with a **?**, the programmer does not understand or support the command.

If a command is sent to the programmer and it fails ("F" is returned), the typical approach would be to then send an "X" command, which returns (and then clears) the errors that have occurred since the last "X" command was sent. The most recent error is returned first.

Exiting Code

To exit CRC, send the **Z** exit command to the programmer. **If you are writing your own driver, the driver's exit command must send a Z followed by a carriage return character to the programmer.**

Reference

Summary of CRC Commands

The CRC commands are ASCII characters (letters or symbols) that are sometimes preceded by alphanumeric entries.

To send a CRC command from your terminal emulator, type the command on the host computer and press **ENTER**.

The command tables are broken up into standard and extended CRC commands. Standard CRC commands are commonly used commands, such as Load, Program, and Verify. Extended CRC commands are more specific device-related commands, such as Set Security Fuse, Fill Fuse Map, and Set Vector Test Options.

Except where noted, the commands use the following notation conventions:

Required command letters are printed in capitals in the summary table and also in the detailed descriptions.

Only uppercase characters can be used.

Valid entries are defined in the detailed descriptions.

Alphanumeric entries that can precede the command are represented by italicized lowercase letters:

h represents a hexadecimal digit.

n represent a decimal digit.

xxx...xxxx represents a string of characters. If the string is a filename, it may use up to 15 total characters: optional drive letter (one character), colon (one character), name (eight characters), a period (one character), and an extension (three characters).

For example, ***nn*02]** indicates that you may precede the **02]** command with two decimal digits.

Summary of Standard CRC Commands

Table 2: Summary of Standard CRC Commands

Command Sent	**Description**	**Response**
& (UniSite w/ SetSite)	Insert Parts Mode	(none)
/ (UniSite w/ SetSite)	View Device Error status	*XXYY*>
-	Invert RAM	>
***hhhhhh*:**	Select device begin address	>
***hhhhhh*;**	Select user data size	>
hhhhhh<	Select memory begin address	>
***nn*=**	Select I/O timeout	>
***nff*A**	Enter translation format	>
B	Blank check	>
C	Compare to port	>
D	Set odd parity	>
E	Set even parity	>
F	Error status inquiry	*HHHHHHHH*>
G	Configuration inquiry	*DD*>
H	No operation	>
I	Input from port	>
J	Set 1 stop bit	>
K	Set 2 stop bits	>
L (*nn*L for SetSite)	Load RAM from device	>
***hh*M**	Enter record size	>
N	Set no parity	>
O	Output to port	>
P (*nn*P for SetSite)	Program device	>
Q	Swap nibbles	>
R (*nn*R for SetSite)	Return status of device	*AAAAA*/*BB*/*C*>
S (*nn*S for SetSite)	View sumcheck	*HHHH*>
T (*nn*T for SetSite)	Illegal-bit test	>
***hh*U**	Set nulls	>
V (*nn*V for SetSite)	Verify device	>
***hhhhhhhh*W**	Set I/O offset	>
X or ***n*X**	Error code inquiry	*HH....HH*>
Y	Display parity errors	*HHHH*>
Z	Exit remote control	(none)
\	Move memory block	>
***hh*^**	Clear/fill RAM with data	>

Summary of Extended CRC Commands

Table 3. Summary of Extended CRC Commands

Command	Description	Response
01]	Display system configuration (responses RRR, II, JJ, VV are for UniSite Only). For example, *SSSS*/*AAAA*/*MM*/*PP*/*KK*/ *QQ*> on the 3900.	*RRR*/*SSSS*/*AAAA*/*MM*/ *PPP*/*IIVV*/*JJVV*/*KKVV*/ *QQ*>
***nn*02]**	Set upload wait time	>
***n*03]**	Set device ID verify option	*HHHHHHHH*> or >
***nn*04]**	Set Remote port baud rate	>
***xxx...xxxx*05]**	Set host command	>
***n*06]**	Select data bits	>
***n*07]**	Set next set member	>
***n*08]** (UniSite w/SetSite)	Set programming mode	>
***nn*09]**	Set the set size	See text following
0A]	Get programmer type	*PP…P*>
0B]	Get type of current device	*HH…H*>
***nn*22]**	Set data word width	>
***n*23]**	Select number of verify passes	>
***n*24]**	Select security fuse programming option	>
***n*26]**	Specify logic verify options	>
***n*27]**	Set/clear enable/disable sec. fuse	>
***n*28]**	Fill fuse map	>
***n*29]**	Set reject count option	>
***hhh*2A] or *hh*2A]**	Enable programming options	>
***hhh*2B] or *hh*2B]**	Disable programming options	>
***nhh*2C]**	Select memory fill option	>
***hh*2D]**	Vector test options	>
***hh*2E]**	Fill RAM with non-repeating test pattern	>
2F] (*nn*2F] for SetSite)	Return 8-character sumcheck	*HHHHHHHH*>
***xxx...xxxx*30]**	Set data file name	>
***n*31]**	Set data source/destination	>
***xxx...xxxx*33]**	Select device manufacturer	>
***xxx...xxxx*34]**	Select device part number	>
***xxx...xxxx*38]**	Load file from disk	>
39]	Delete all RAM files	>
***xxx...xxxx*3B]**	Delete disk file	>
***n*3C]**	Set data transfer port	>
***xxx...xxxx*3E]**	Select Keep Current algorithm	>
***n*40]**	Upload device information	See text following
***n*41]**	Perform self-test and upload results	*AAA...AA*>
43]	Upload yield tally	See text following
45]	High-speed download	See text following
46]	Clear yield tally	>
49]	Suspend remote (CRC) mode	Displays terminal screen
***n*4A]**	Get filename from disk	*AAA...AA*>
***n*4D]**	Select algorithm type	>

Command	Description	Response
***n*4F]**	Set RAM device selection	>
***n*52]**	Select media for algorithms (floppy disk or MSM)	>
***xxx...xxxx*53]**	Save RAM data to disk file	See text following
54]	Upload device footnote	See text following
55]	Upload device-specific message	See text following
56] (*nn*56] for SetSite)	Upload memory verify failure	*ddPAAAAAAAAHHhh*>
57] (*nn*57] for SetSite)	Get checksum of operation	See text following
58]	Upload system ID	*HHHH HHHH HHHH*>
***n*59]** (AutoSite only)	Enable/disable capacitor configuration test	>
5A]	Display list of parameters	See text following
5B]	Clear vector data	>
5C]	Load system files for Custom Menu (CM) algorithm disk	>
5D]	Write system files to CM disk	>
5E]	Write algorithms to CM disk	>
***n*5F]**	Select the CM algorithm drive for creating CM algorithms	>
60]	Get number of sectors	dd>
***n*61]**	Get sector configuration settings	HHHH HHHH>
***nhhhhhhhh*62]**	Set sector configuration settings	>
63]	Reboot programmer	
***xxx...xxxx*64]**	Select device part number for CM (use *xxx.xxxx*33] to select the manufacturer)	
***A*65]** (2900/3900)	Return software version number	*X.XX*
66]	Set abort on empty socket	
67]	Set checksum word size (8 or device width)	
***n*67]**	Set checksum wordsize: 0 = 4 bit 1 = 8 bit 2 = device width	
68]	Device operations supported	> (Hexadecimal number)
69]	Display pin adapter information	See text following
6C]	MSM test	See text following
***n*6D]**	Set checksum type	>
***n*70]**	Get/test device electronic ID	See text following
71]	Stand-alone device erase	>
***hh…h*72]**	Save system file	>
***hh…h*73]**	Save algorithm file	>
77]	Switch to parallel port	See text following
80]	Echo test	See text following
A7]	Swap bytes	>
C1]	Diagnostic looping	

Command	**Description**	**Response**
DC]	Device check	See text following
DF] (UniSite w/SetSite)	View status of sockets	See text following
EB]	Input JEDEC data from host	>
EC]	Output JEDEC data to host	>
FC]	Restore CRC entry default parameters	>
FD]	Restore user-defined CRC parameters	>
FE]	Save user-defined CRC parameters	>

Description of CRC Commands

The following section describes the available CRC commands in more detail than the "Summary of CRC Commands" section.

Control System Commands

The following paragraphs describe how to discontinue an operation or exit remote mode. These commands control the programmer rather than performing a specific programming-related operation (such as blank testing or verifying devices).

No Operation (H)

Use the **No Operation** (ASCII character H) command to verify that communications have been established between the programmer and remote computer (or terminal). Send an **H** command to the programmer immediately after CRC has been entered. A prompt character (>) is returned if communications have been established correctly between the programmer and remote computer (or terminal).

Starting/Stopping Commands

Table 4 lists and describes the commands used to start or stop CRC operations.

Table 4. Computer Remote Control Commands

ASCII	Hex	Description
ENTER or RETURN	0D	**Carriage Return —** Use ENTER from your PC to tell the programmer to run a command. All commands (except ESC and BREAK) are ignored if not followed by ENTER.
ESC	1B	**Escape —** Use ESC from your PC to tell the programmer to unconditionally discontinue any operation in progress and send the prompt character, >. The programmer waits for further instructions.
BREAK		**Break —** (Serial port only) Press BREAK from your PC to tell the programmer to unconditionally discontinue any operation in progress and send the prompt character, >. A BREAK is the equivalent of an intentional framing error, holding the serial line active for 2 or 3 character widths.
DIR		**Direction Change —** (Parallel port only) Changing the state of the direction line during a data transfer operation (sending or receiving data) tells the programmer to unconditionally discontinue the operation in progress.

Exiting Remote Mode

To exit CRC, send the **Z ENTER** command to the programmer. If you are writing your own driver, the driver's exit command must send a **Z ENTER** (Z followed by the carriage return character) to the programmer.

Description of Standard CRC Commands

The Standard CRC Commands are described in this section. Note that the lowercase letters preceding the commands are arguments that must be specified according to the options listed under the corresponding command description. Except where noted, the standard CRC commands use the following notation conventions:

Lowercase letters indicate arguments that must be specified

h represents a hexadecimal digit.

n represents a decimal digit.

xxx...xxxx represents a string of characters or a filename. Filenames are limited to 14 total characters in length: drive specification (2 characters) plus name (8 characters) plus extension (3 characters preceded by a dot). For example, ***nn*02]** indicates that you may precede the **02]** command with two decimal digits.

ASCII	Description
& (UniSite w/SetSite)	**Insert Parts Mode** Puts a UniSite programmer in a wait state, allowing you time to insert the devices. UniSite will remain in this state until you push the SetSite socket lever forward to the Start position to begin a device operation. This command also clears the device statistics, but does not affect the yield tally data. The & command may be halted by pressing ESC or pulling SetSite's socket lever to the Open (fully back) position.
/ (UniSite w/SetSite)	**View Device Error Status** Returns the results of the previous device operation(s). The results are returned as a 4-character string in the form *XXYY*, where *XX* is the number of devices that did not program successfully and *YY* is the total number of devices you attempted to program.
-	**Invert RAM** Inverts the data in RAM within the address range defined by the Beginning Memory Address and the Memory Block Size. If the User Data Size is set to 0, all of User Memory is inverted.
hhhhhh:	**Select Device Begin Address** Sets the first device address to load, program, or verify. This command is also used as the destination address in a RAM to RAM block move. The Device Begin Address defaults to 0 if no address precedes the colon.
hhhhhh;	**Set User Data and Device Block Size** Sets the number of bytes to be uploaded, downloaded, transferred, or programmed. This command sets both the User Data Size and the Device Block Size. If *hhhhhh*; is greater than zero, the User Data Size and the Device Block Size are set to *hhhhhh* unless *hhhhhh* is greater than the physical size of the device, in which case the Device Block Size is set to the physical size of the device. If *hhhhhh*; is zero (0;), the Device Block Size is set to the physical size of the device, and the User Data Size is set to cover the entire device. For example, for a 16-bit device of 4000 16-bit words in size, Device Block Size is set to 4000, and User Data Size is set to 8000. If no argument precedes the semicolon (";" by itself), the User Data Size is set to the size of User Memory, and the Device Block Size is set to the size of the currently selected device.
hhhhhh<	**Select Memory Begin Address** Sets the first RAM address from which or to which data will be transferred. This address is also used as the Begin RAM Address where the programming data is located. The Memory Begin Address defaults to 0 if no address precedes the <.

ASCII	Description
***nn*=**	**Select I/O Timeout** Specifies the number of seconds the programmer will wait during a download before it returns an I/O timeout error (CRC error code 46). Valid arguments range from 01 to 99 seconds. To disable the I/O Timeout, either specify an I/O Timeout of 00 seconds, or send a null value (i.e., just send the = command). The I/O Timeout defaults to 30 seconds.
***nff*A**	**Enter Translation Format** Selects the instrument control code *n* and the data translation format *ff* to be used for I/O data transfers through the Remote port. If 1 or 2 digits precede the **A**, the digits select the data translation format and the instrument control code defaults to 0. If 3 digits precede the **A**, the first digit designates the instrument control code and the last two digits specify the data translation format. For example, sending **191A** selects instrument control code 1 and data translation format 91. The data translation formats and the instrument control codes are described in the Translation Formats section in your **Programmer's User Manual**.
B	**Blank Check** Performs a blank check on the currently socketed device.
C	**Compare to Port** Compare data in the programmer's RAM with data received through the Remote port using the current data translation format. (JEDEC format cannot be used with this command: this command works only for memory formats.) The current Memory Begin Address and I/O Offset are used to calculate the RAM address where the data is located to compare against the incoming data.
D	**Set Odd Parity** Sets odd parity for serial data transfers through the Remote port.
E	**Set Even Parity** Sets even parity for serial data transfers through the Remote port.
F	**Error Status Inquiry** Returns a 32-bit number in the format HHHHHHHH, where each H is a hex character. The 32-bit word defines the accumulated errors in the error status word since the last **F** command. See the section titled "Error Status Word" in this chapter for more information.
G	**Configuration Inquiry** Returns the configuration information in the form DD, where DD is the disk version. For example, if 13 is returned, the disk version is 1.3. Additional configuration information can be obtained by using the extended command **01]**.
H	**No Operation** Returns the > prompt followed by a **RETURN** (ENTER) and, if specified, a line feed. No operation is performed.
I	**Input From Port** Instructs the programmer to accept formatted data from the Remote port using the current data translation format and load that data into RAM. For memory devices, the current Memory Begin Address and I/O Offset values are used to calculate the RAM address where the input data is loaded. For logic devices, all of the fuse map and structured vectors will be received and placed in RAM at the appropriate address. You must select the JEDEC format if a logic device is selected. An XOFF is sent to the host computer after the **I** command is received. This allows the programmer time to get ready to receive data from the host computer. An XON is automatically sent to the host computer to begin the data transfer if the instrument control code is not 1. (This is done only for remote mode.) Even if your host system has hardware handshake or XON/XOFF, we recommend you provide a 20 millisecond delay between the time you send the **I** command and the first byte of data. If your system does not have hardware handshake or XON/XOFF capability, you must provide the delay to separate the **I** command from the first byte of data sent. A delay of 1/2 second to 8 seconds is suggested, depending on whether you use the Fill Memory option and the size of your User Memory.

ASCII	Description
J	**Set 1 Stop Bit** Sets one stop bit for serial data transfers through the Remote port.
K	**Set 2 Stop Bits** Sets two stop bits for serial data transfers through the Remote port.
L (***nn*L** for SetSite)	**Load RAM From Device** Loads data from the currently selected device into programmer RAM. For logic devices, the entire device is loaded. For memory devices, specify the following parameters before you send this command: First device address copied from (Device Begin Address), First RAM address copied to (Memory Begin Address), and Size of the block copied (Memory Block Size) For SetSite operation, send *nn*L, where *nn* specifies which device socket to load from. Valid arguments for *nn* range from 01 to 08.
***hh*M**	**Enter Record Size** Sets the number of data bytes per record for serial data transfers.
N	**Set No Parity** Disables parity checking for serial data transfers through the Remote port.
O	**Output To Port** Instructs the programmer to output formatted data to the Remote port using the current data translation format. For memory devices, the current parameter settings for Memory Block Size, Memory Begin Address, and I/O Address Offset are used. The complete fuse map and structured vectors are output for logic devices. The data translation format must be JEDEC if a logic device is selected.
P (***nn*P** for SetSite)	**Program Device** Programs a device with data in the programmer's RAM. For logic devices, the entire device is programmed. For memory devices, specify the following parameters before you send the **P** command: First address to program from (Memory Begin Address), Number of bytes to program (Memory Block Size) and First device address to program (Device Block Size). For SetSite operation, send *nn*P, where *nn* is the number of devices in the set you want to program. Valid arguments for *nn* range from 01 to 08.
Q	**Swap Nibbles** Swaps the high-order and low-order nibbles in a given memory range. Use the *hhhhhh*< and *hhhhhh*; commands to specify the memory begin address and the size of the memory range to swap. The memory begin address (specified by the *hhhhhhh*< command) added to the block size (specified by the *hhhhhh*; command) cannot exceed the size of user memory. Entering a block size of 0 will swap all memory beginning with the address specified by the *hhhhhhh*< command. Use the **A7]** command if you want to swap bytes. *Note: This command will not work if you have a logic device selected.*
R (***nn*R** for SetSite)	**Return Status Of Device** Returns the attributes of the selected device. Data is output in the form *aaaaa/bb/c*. For memory devices, *aaaaa* indicates the device's word limit in hex, *bb* is the word size in decimal, and *c* = 0 (VOL) or 1 (VOH). For logic devices, *aaaaa* indicates the number of fuses and *bb* is the number of device pins. For SetSite operation, send *nn*R, where *nn* specifies which device socket to return status about. Valid arguments for *nn* range from 01 to 08.

ASCII	Description
S (***nn*S** for SetSite)	**View Sumcheck** Returns the sumcheck of the RAM data as a 4-digit hex number. For memory devices, sumcheck starts at the beginning of User RAM and continues for the word limit (device size) of the selected device. For logic devices, sumcheck starts at the beginning of User RAM plus 8 bytes and continues for the device size divided by 8. For SetSite operation, send *nn*S, where *nn* specifies the device socket to sumcheck. Valid arguments for *nn* range from 01 to 08.
T (***nn*T** for SetSite)	**Illegal-bit Test** Tests the selected device for illegal bits. An illegal bit is defined as a programmed bit in the device that does not exist in RAM. For SetSite operation, send *nn*T, where *nn* specifies the number of devices in the set. Valid arguments for *nn* range from 01 to 08.
***hh*U**	**Set Nulls** Sets the number of nulls after a carriage return on output data transfer operations. This command also enables/disables sending of a line feed after every carriage return sent out (for responses too). If the argument is FF, no line feeds or nulls will be sent after each carriage return. The number of nulls defaults to zero and line feeds are enabled if no argument precedes the **U**.
V (***nn*V** for SetSite)	**Verify Device** Verifies the data in the programmer's RAM against the data in the socketed device. For logic devices, the entire device is verified. For memory devices, the Memory Begin Address, Device Begin Address, and Memory Block Size may be set prior to sending this command. For SetSite operation, send *nn*V, where *nn* specifies the device socket to verify. Valid arguments for *nn* range from 01 to 08
***hhhhhhhh*W**	**Set I/O Offset** Sets the I/O Offset Address to be used in I/O operations. If FFFFFFFF precedes the **W** command, the I/O Offset defaults to 0 for output operations and the first incoming address for input operations. The I/O Offset defaults to 0 if no argument precedes the **W**. For input operations, the address where the data is placed is calculated by subtracting the I/O Offset from the incoming address and adding to it the Memory Begin Address. For output operations, the outgoing address is calculated by taking the address where the data is located, subtracting the Memory Begin Address and adding the I/O Offset.
X or *n*X	**Error Code Inquiry** Returns the last 20 error codes and clears them from memory. Each error code is returned as a 2-digit hex character. See the section titled "CRC Error Codes" later in this chapter for explanations of the CRC error codes. A normal prompt > and a carriage return are returned if no errors have occurred. Normally, the **X** command is sent if the programmer returns the F error code, which means that the previous command failed. The ***n*X** CRC operates the same as the X command if *n* is **0**. If n is **1**, the accumulated warning codes are returned and the running count of the warning codes is reset to 0.
Y	**Display Parity Errors** Returns the number of parity errors as a 4-digit hex number and clears the parity error counter. The parity error counter is also cleared at power on, when a **Y** command is sent, or when a parity command (**D**, **E**, **N**) is sent.
Z	**Exit Remote Control** Exits computer remote control and returns control to the programmer's terminal interface.
[	**View Device Family/Pinout Code** Returns the family/pinout code of the currently selected device. The family and pinout codes are returned in the form *fffppp* where *fff* is the 3-digit family code and *ppp* is the 3-digit pinout code.

ASCII	Description
\	**Move Memory Block** Moves data from one RAM location to another. The Memory Begin Address, Device Begin Address, and Memory Block Size all should be set prior to execution of this command. The Memory Begin Address is the source address. The Device Begin Address is the destination address. The Memory Block Size determines the number of bytes to move. Block size defaults to the size of user memory if the Memory Block Size is set to 0.
***hh*^**	**Clear/Fill RAM With Data** Fills every address within the range defined by the Memory Begin Address and Memory Block Size with the specified argument. The Memory Begin Address and Memory Block Size should be set prior to execution of this command. If you set the Memory Block Size to 0, all of User RAM will be filled with the data pattern, which effectively wipes out any data you had stored in RAM. If you send ^ without an argument, all of User RAM will be cleared (filled with 00) regardless of how you specified the block parameters.

Description of Extended CRC Commands

The Extended CRC Commands are described in this section. As with the Standard CRC Commands, the description of commands uses the following notation conventions:

Lowercase letters indicate arguments that must be specified

h represents a hexadecimal digit.

n represents a decimal digit.

xxx...xxxx represents a string of characters or a filename. Filenames are limited to 14 total characters in length: drive specification (2 characters), plus name (8 characters), plus extension (3 characters preceded by a dot).

For example, ***nn*02]** indicates that you may precede the **02]** command with two decimal digits.

ASCII	Description
01]	**Display System Configuration** Returns system configuration

```
RRR/SSSS/AAAA/MM/PPP/IIVV/JJVV/KKVV/QQ>
```

where:

RRR is the current ROM version (UniSite only)
SSSS is the current system version
AAAA is the Algorithm version
MM is the decimal number of 64K-byte banks of RAM memory available
PPP is the decimal number of pin drivers available
II identifies which PSM (small module) is installed (UniSite only):

***II* Value**	**Module**
00	no module is installed
01	Site 40
02	Site 48
03	Site48HS

VV is the version number on UniSite only.
JJ identifies which FSM (large module) is installed (UniSite only):

***JJ* Value**	**Module**
00	no module is installed
01	ChipSite
02	SetSite
04	PinSite
05	USM-340

VV is the version number on UniSite only.

KK is a hexadecimal number identifying which Base or module is installed. *KK* has the following values for Bases:

***KK* Value**	**2900/3900**	**UniSite**	**AutoSite/2500**
00	No Base detected	No Base detected	No prog. module detected
01	40-pin DIP Base	PLCC/LCC Base	---
02	PLCC Base	PGA Base (version 1)	---
03	---	SOIC Base (version 1)	300-mil DIP
04	SOIC Base	PGA PSBASE-0402 base (PGA base version 2)	---
05	48-pin DIP Base	PPI Base	48-pin DIP Base
06	---	SOIC PSBASE-0302 Base (SOIC Base version 2)	---
07	---	---	20-pin PLCC
08	---	---	28-pin PLCC
09	---	---	32-pin PLCC
0A	---	---	44-pin PLCC
10	PLCC Base	---	PLCC Base
11	PGA Base	---	---
12	---	---	52-pin PLCC
13	---	---	68-pin PLCC
14	---	---	84-pin PLCC
15	---	---	600-mil DIP (PM 3000 and 7000 only)
16	---	---	600-mil DIP (PM2000 only)
60	---	---	SOIC (300- and 450-mil)
61	---	---	SOIC (350- and 530-mil)
62	---	---	300-mil DIP (new version)
63	---	---	600-mil DIP (new version)
C0	PPI Base (3900)	---	---
E0	PPI Base (2900)	---	---

VV is the version number on UniSite only.

QQ is the decimal number of the PPI Adapter installed (if *QQ* is 00, no PPI Adapter is installed)

ASCII **Description**

***nn*02]** **Set Upload Wait Time** Specifies the number of seconds the programmer will wait before uploading data. Valid arguments range from 00 to 99 seconds.

***n*03]** **Select Electronic ID** Enables/disables the electronic ID test and also can return the Electronic ID of the selected device. The argument must be one of 0, 1, or 2, where

0	Disables electronic ID.
1	Enables electronic ID.
2	Returns electronic ID as eight hex digits in the form *hhhhhhhh*.

Leading zeros are sent for those devices without an 8-digit ID (such as 0000890D).

***nn*04]** **Set Remote Port Baud Rate or Set Handler Port Baud Rate** Sets the baud rate for the Remote (Handler) port. Valid arguments are listed and described below:

nn	Baud Rate	*nn*	Baud Rate
01	50	10	1500
02	75	11	1800
03	110	12	2000
04	134.5	13	2400
05	150	14	4800
06	200	15	7200
07	300	16	9600
08	600	17	19.2K
09	1200		

***xxx...xxxx*05]** **Set Host Command** Sets the command string to be sent to the host for upload or download of data. Valid arguments can range in length from 0 to 58 characters. If no argument precedes the **05]** command, no host command is sent. The programmer appends a carriage return to the end of the string.

***n*06]** **Select Data Bits** Sets the number of data bits for serial data transfers through the Remote port. Valid arguments are listed and described below:

8	Selects 8 data bits.
7	Selects 7 data bits.

***n*07]** **Set Next Set Member** Determines data organization during serial set programming. For example, if you are programming 4-bit devices and have set the Data Word Width to 8, setting the Next Set Member (*n*=2) means that the upper 4 bits of each data byte (rather than the lower 4 bits) are used to program the device.

***n*08]** (UniSite w/ SetSite) **Set Programming Mode** Selects the programming mode. Valid arguments are listed and described below:

0	Selects single device mode.
1	Selects gang/set mode.

ASCII	Description
0A]	**Get Programmer Type** Returns the type of the programmer and whether or not it has the hard drive/MSM. It returns one of the following strings: UniSite UniSite_MSM 2900 2900_MSM 3900 3900_MSM AutoSite AutoSite_MSM A **3980** programmer will return **3900_MSM**. Also, no **2900** programmer has an MSM, but the return is shown above in the interest of completeness. There is currently no method of discerning from remote mode if the programmer is an *xpi* model. The only indication available is that if it does not have an MSM, then it is not an *xpi*.
0B]	**Get Type of Current Device** Returns a hexadecimal number that gives an indication of the category of device of the currently selected part. It returns a hexadecimal number that can take on any of the following values: 1 EPROM 2 Electrically erasable PROM 3 TTL PAL 4 Bipolar PROM device 5 AIM PROM device 6 Electrically erasable PAL device 7 Microcontroller with EPROM 8 POF PAL device 4002 Byte erasable EEPROM 9 Static RAM device - backwards device and power-up diff. A POF device type which supports packet #17 B LOF device type C Same as POF_17 but Electrically Erasable D Some ACTEL FPGA devices
***nn*22]**	**Set Data Word Width** Specifies, in bits, the width of a data word in the device being programmed. Valid arguments range from 4 to 64.

ASCII	Description
***n*23]**	**Select Number of Verify Passes** Selects the number of verify passes and the type(s) of voltage(s) used during a verify operation. Valid arguments are listed and described below: 0 Specifies no verify passes. 1 Performs a single-pass verify with nominal V_{CC}. 2 Performs a two-pass verify, one at the maximum allowed V_{CC} and one at the minimum allowed V_{CC} value.
***n*24]**	**Enable Security Fuse** Enables/disables programming the security fuse(s). Valid arguments are listed and described below: 0 Disables programming the security fuse(s). 1 Enables programming the security fuse(s).
***n*26]**	**Specify Logic Verify Options** Selects the type of logic verification to perform during a verify operation. Valid arguments are listed and described below: 0 Performs the fuse verify test followed by a structured vector test. 1 Performs only the fuse verify test. 2 Performs the structured vector test. *Note: The 2900 does not support vector testing for logic devices with more than 44 pins, regardless of the verify option setting. The 3900, AutoSite, 2500, and UniSite do not support vector testing for logic devices with more than 84 pins, regardless of the verify option setting.*
***n*27]**	**Set/Clear Enable/Disable Security Fuse** Enables/disables programming of the security fuse and sets the state of the security fuse. Valid arguments are listed and described below: 0 Disables programming of all security fuses and sets all security fuse states in RAM to 0. 1 Disables programming of all security fuses and sets all security fuse states in RAM to 1. 2 Enables programming of all security fuses and sets all security fuse states in RAM to 0. 3 Enables programming of all security fuses and sets all security fuse states in RAM to 1. 4 Sets first security fuse state in RAM to 1. 5 Sets second security fuse state in RAM to 1. 6 Sets third security fuse state in RAM to 1.
***n*28]**	**Fill Fuse Map** Specifies the fuse state with which to fill the fuse map. Valid arguments are listed and described below: 0 Fills the fuse map in RAM with 0s. 1 Fills the fuse map in RAM with 1s.
***n*29]**	**Set Reject Count Option** Selects the maximum number of programming pulses required to program a device before the programmer rejects the device as unprogrammable. Valid arguments are listed and described below: 0 Selects the number of programming pulses specified by the device manufacturer. 1 Selects a single programming pulse or military reject count.

ASCII	Description
***hhh*2A]**	**Enable Programming Options** Enables one or more programming options. The argument can be a 2- or 3-digit hex number. Valid arguments are listed below: Bit 0 (hex 01) = enable illegal bit check Bit 1 (hex 02) = enable blank check Bit 2 (hex 04) = enable yield tally Bit 3 (hex 08) = enable erase EE device Bit 4 (hex 10) = enable odd/even byte swap Bit 5 (hex 20) = enable JEDEC I/O translate DIP/LCC Bit 6 (hex 40) = enable continuity check Bit 8 (hex 100) = enable special data switch #1 (optional) Bit 9 (hex 200) = enable special data switch #2 (optional)
***hhh*2B]**	**Disable Programming Options** Disables one or more programming options. The argument can be a 2- or 3-digit hex number. Valid arguments are the same as listed for the **2A]** command.
***nhh*2C]**	**Select Memory Fill Option** Specifies what data User RAM will be filled with before a download begins. User RAM will be filled with a 2-digit hex number (the *hh* argument) when the *n* argument is 2. Valid arguments for *n* are listed and described below: 0 Memory is not changed. 1 Default (unused locations are initialized to the unprogrammed state for the device type selected) 2 Fill unused memory locations with the specified 2-digit hex number.
***hh*2D]**	**Vector Test Options** Enables or disables the compensated vector test, serial vector test, and high speed logic driver options. Valid arguments are listed and described below: Bit 0 = 0 to disable compensated vector test Bit 0 = 1 to enable compensated vector test Bit 1 = 0 to disable high speed logic driver Bit 1 = 1 to enable high speed logic driver Bit 2 = 0 to disable serial vector test Bit 2 = 1 to enable serial vector test
***hh*2E]**	**Fill RAM With Non-Repeating Test Pattern** Fills RAM between block limits with a test pattern that does not repeat on address boundaries. The first byte of the pattern will be the value of the two-digit hexadecimal argument.
2F]	**View 8-Character Sumcheck** Returns the 8-character hexadecimal sumcheck of the data in User RAM. Refer to the CRC **S** command for more information.
***xxx...xxxx*30]**	**Set Data File Name** Sets the filename for any subsequent file operations.
***n*31]**	**Set Data Source/Destination** Sets the source/destination for a data file. Valid arguments are listed and described below: 0 RAM 1 Disk

ASCII	Description
***xxx*...*xxxx*33]**	**Select Device Manufacturer** Selects the device manufacturer for device operations and when adding a device to a Custom Menu. Valid arguments can range from 1 to 13 alphanumeric characters. Valid arguments must also match the manufacturer name exactly as it appears on the Manufacturer List screen, or as it is uploaded via the **40]** command. The manufacturer selected does not take effect until the **34]** command or **64]** (for Custom Menu algorithms) is sent to select the device part number.
***xxx*...*xxxx*34]**	**Select Device Part Number** Selects the device part number for device operations. Valid arguments can range from 1 to 29 alphanumeric characters. Valid arguments must also match the part number as it appears on the Parts Number screen for the selected Manufacturer, or as it is uploaded via the **40]** command. This command selects an algorithm based on the part number sent in this command and the Manufacturer sent in the **33]** command.
***xxx*...*xxxx*38]**	**Load File From Disk** Loads a disk file into RAM. Valid arguments range from 1 to 15 characters. The entire file is always loaded, and the User Data Size is updated to reflect the size of the file loaded into RAM.
39]	**Delete All RAM Files** Clears all files stored in User RAM.
***xxx*...*xxxx*3B]**	**Delete Disk File** Deletes a disk file. Valid arguments range from 1 to 15 characters and may include the * wildcard character. For example, to delete the file u1.dat, send the following command: **U1.DAT3B]**.
***n*3C]**	**Set Data Transfer Port** Specifies which port (Terminal or Remote) the programmer will use for CRC data transfer operations (such as the input, output, and JEDEC input/output commands). Unless specified otherwise, the programmer defaults to the Remote port for data transfer operations. This command makes it possible to transfer data to the programmer from a system other than the one currently running your CRC driver program. The driver program would be communicating with the Remote port of the programmer and could initiate a download or upload with a different computer connected to the Terminal port of the programmer. This is useful if the data files you want to use exist on a system other than the one running your CRC driver program. CRC commands are still recognized only on the Remote port. Valid arguments are listed and described below: 0 Remote Port; 1 Terminal Port
***xxx*...*xxxx*3E]**	**Select Keep Current or Custom Menu algorithm** The Keep Current or Custom Menu algorithm is loaded from the specified .KC*x* or .CM*x* filename. With this command, Keep Current and Custom Menu algorithms with different revision numbers may be selected for the same device.

ASCII **Description**

***n*40]**

Upload Device Information Depending on the argument supplied, the *n***40]** command uploads either a terse list of all the supported devices, information about the currently selected device, or a verbose list of all the supported devices. Valid arguments are listed and described below:

0	upload terse device list
1	upload current device information
2	upload verbose device list

Note: ***The*** **040]** and **240]** commands (the terse and verbose lists of devices supported) share the same format. Certain fields in the terse list are zeroed out, making the upload quicker.

While the **040]** and **240]** commands yield results that appear to be insignificantly different, the two commands operate in ways that can affect the performance of a CRC driver.

The **240]** command needs to access data stored on the Algorithm disk(s) to build the data stream it returns. This means that the upload of device information could take longer than desired.

On the other hand, the **040]** command does not need to access data on the Algorithm disk. As a result, the **040]** command provides a quicker method of uploading device support information.

The data fields left blank by the **040]** command   the Electronic ID field and the Module Support field   should not be needed in most CRC driver applications since that information is not required for device operations. However, if you need the data in these fields, you can obtain it in one of two ways. You can use the **240]** command to upload the full set of device support information. Or, you can use the **040]** command to upload the terse set of device support information and then use the **140]** command to get information for a particular device after selecting the device.

ASCII	Description
*n*40]	**Upload Device Information (continued)**

040] **Upload Terse Device List** The **040]** command uploads a terse list of all the devices supported by the programmer. The only difference between the **040]** and the **240]** commands is that some fields in the data stream returned by the **040]** command are zeroed out. The zeroed out fields are represented below as "Zeroes". The data is transferred as a string of characters in the following format:

Definition	**Number of Bytes**
Number of manufacturers	2
<CR><LF>	2 hex
Next is data for EACH device manufacturer, organized as follows:	
Device manufacturer's name	1 to 32
Colon	1
Number of devices for this manufacturer	3
Next, the following is repeated for each device this manufacturer supports:	
<CR><LF>	2 (hex)
Device's part number	1 to 32
Colon	1
Family code	4
Pinout code	4
Zeroes	8
Zeroes	2
<CR><LF> next device for *this* manufacturer . . . etc.	
<CR><LF> next manufacturer . . . etc.	

140] **Upload Current Device Information** The **140]** command uploads information about the currently selected device. If the device has been selected by family/pinout code, the silicon signature is set to 0. The current part information is transferred as a string of characters in the format described below.

Definition	**Number of Bytes**
Device manufacturer's name	1 to 32
Colon	1
Device's part number	1 to 32
Colon	1
Family code	4
Pinout code	4
Electronic ID	8
Unused	2

ASCII **Description**

***n*40]** **Upload Device Information (continued)**

240] **Upload Verbose Device List** The **240]** command uploads a verbose list of all the devices supported by the programmer. The only difference between the **040]** and the **240]** commands is that some fields in the data stream returned by the **040]** command are zeroed out. The data is transferred as a string of characters, in the following format:

Definition	**Number of Bytes**
Number of manufacturers	2
<CR><LF>	2 hex

Next is data for EACH device manufacturer, organized as follows:

Device manufacturer's name	1 to 32
Colon	1
Number of devices for this manufacturer	3

Next, the following is repeated for each device this manufacturer supports:

<CR><LF>	2 (hex)
Device's part number	1 to 32
Colon	1
Family code	4
Pinout code	4
Electronic ID	8
unused, reserved field	2

<CR><LF> next device for *this* manufacturer . . . etc.
<CR><LF> next manufacturer . . . etc.

***n*41]** **Upload Self-test Results** performs the following self-tests and reports the results:

141]	Calibration test
241]	PCU test
341]	FEPROM test
441]	Serial ports test
541]	System RAM test
641]	User RAM test
*A*741]	Floppy disk test, where *A* is the name of the drive to be tested (the drive character is not case sensitive)
841]	Base/adapter/relays test
941]	Pin driver board test

All these tests can be aborted in remote mode with an ESC command. An aborted test returns an “F” with error code 0xFF.

ASCII	**Description**
***n*41]**	**Upload Self-test Results (continued)**

041] Returns the results of the previous self-test as a 12- (2900), 13- (2500, 3900, AutoSite), or 30-character (UniSite) string in which each character represents the results of a different test. The tests and their positions in the string are described below. The 2900 returns a 12-character string in the following format:

Character Position	**Item Tested**	**Character Position**	**Item Tested**
1	Spare	7	Disk drive
2	EPROM	8	Option board
3	System RAM	9	Pin driver board #1
4	User RAM	10	Calibration
5	Serial port A	11	PCU (Pin Control Unit)
6	Serial port B	12	Base

The 2500, 3900, and AutoSite return a 13-character string in the following format:

Character Position	**Item Tested**	**Character Position**	**Item Tested**
1	Spare	7	Disk drive
2	EPROM	8	Option board
3	System RAM	9	Pin driver board #1
4	User RAM	10	Pin driver board #2
5	Serial port A	11	Calibration
6	Serial port B	12	PCU
		13	Base or module

UniSite returns a 30-character string in the following format:

Character Position	**Item Tested**	**Character Position**	**Item Tested**
1	Spare	16	Pin driver board #7
2	EPROM	17	Pin driver board #8
3	System RAM	18	Pin driver board #9
4	User RAM	19	Pin driver board #10
5	Serial port A	20	Pin driver board #11
6	Serial port B	21	Pin driver board #12
7	Disk drive A	22	Pin driver board #13
8	Disk drive B	23	Pin driver board #14
9	Mass Storage Module	24	Pin driver board #15
10	Pin driver board #1	25	Pin driver board #16
11	Pin driver board #2	26	Pin driver board #17
12	Pin driver board #3	27	Waveform generator bd.
13	Pin driver board #4	28	PCU
14	Pin driver board #5	29	PSM board
15	Pin driver board #6	30	FSM board

Each test can produce one of four result codes, which are described below:

Code	**Definition**
-	Hardware not installed
?	Untested
F	Failed self-test
P	Passed self-test

ASCII	Description
43]	**Upload Yield Tally** Uploads the yield tally for up to sixteen different devices. The yield tally is uploaded in the following format (one line is returned for every device entry in the yield tally statistics file):

Manufacturer's name or family/pinout	25 characters
Total parts attempted	5 characters
Space	1 character
Total parts passed	5 characters
Space	1 character
Total illegal bit/blank errors	5 characters
Space	1 character
Total verify errors	5 characters
Space	1 character
Total structured test errors	5 characters
Space	1 character
Total program failures	5 characters
Carriage return, line feed	2 characters

46] **Clear Yield Tally** Clears the yield tally statistics.

49] **Suspend Remote Mode** Suspends remote mode temporarily and returns to Terminal mode, where menu data will be sent to the port specified by the User Menu Port parameter. The values for all system parameters will still contain the values they had while in remote mode prior to the **49]** command. Any changes to the parameters will apply to remote mode when remote mode is resumed.

Note: The programmer will send a prompt back before suspending remote mode. If neither of the serial ports is connected, however, an error DS will result.

The **49]** command allows you to temporarily leave remote mode, perform some operations and then re-enter remote mode with the system parameters unchanged. For example, the following scenario would be possible with the **49]** command:

1. Enter remote mode.
2. Select a device manufacturer (***xxx...xxxx*****33]** command).
3. Select a device part number (***xxx...xxxx*****34]** command).
4. Change the setting of some programming parameters, such as illegal bit check, blank check, etc. (***hhh*****2A]** command).
5. Suspend remote mode and return to Terminal mode (**49]** command).
6. Perform terminal functions, such as viewing a fuse pattern or editing memory.
7. Re-enter remote mode. (At this point, the changes to the programming parameters mentioned above would still be in place.)
8. Program the device.

*Note: The **49]** command differs from the **Z** command (Exit Remote Mode). The **Z** command exits CRC and sets the system parameters to the values they had before remote mode was used. If you enter remote mode after exiting remote mode with the **Z** command, the system parameters are set to CRC default values.*

ASCII **Description**

***n*4A]**

Get Filename From Disk Displays the current filename or scrolls backwards or forwards through the filenames of the files found in the disk drive(s). Valid arguments are listed and described below:

0 uploads the filename of the next file in the directory list
1 uploads the filename of the previous file in the directory list
2 rebuilds the directory list and uploads the filename of the first file

The ***n*4A]** command is designed to be used as a front end to the ***xxx...xxx*38]** command. Use the ***n*4A]** command to get a filename which can then be sent with the ***xxx...xxx*38]** command to load the file from disk.

***n*4D]**

Select Algorithm Type Selects the set of algorithms to use with the ***xxx...xxxx*33]**, ***xxx...xxxx*34]**, ***n*40]**, and **@** commands. Valid arguments for *n* are described below:

0 Use the set of algorithms included on the Algorithm disks.
1 Use the extended set of algorithms (if any are available) or algorithms from an Archive disk. Devices are selected from the **alg.ext** file.
2 Use the Keep Current set of algorithms (if any are available). The Keep Current algorithms are downloaded from the Keep Current BBS.
3 Use a Custom Menu set of algorithms (if any are available).

The ***n*4D]** command lets you switch between algorithms included on the Algorithm disk and, for example, a collection of Keep Current algorithms. Consider this scenario:

1. Enter remote mode.
2. Select the standard device file (***0*4D]** command).
3. Select a device manufacturer (***xxx...xxxx*33]** command).
4. Select a device part number (***xxx...xxxx*34]** command).
5. Program the device (**P** command).
6. Select the Extended algorithm file (***1*4D]** command).
7. Select a device manufacturer (***xxx...xxxx*33]** command).
8. Select a device part number (***xxx...xxxx*34]** command).
9. Program the device with the Extended algorithm (**P** command).

This example shows how to use ***n*4D]** command to select a different algorithm source.

***n*4F]**

RAM Device Selection Enables/disables RAM device selection. Valid arguments are listed and described below:

0 Disable RAM device selection
1 Enable RAM device selection; allowed only if the programmer has at least 4 MB of memory.

The 14F] command instructs the programmer to select the device algorithm from a special area 1 MB in size located at the high end of user RAM. After the algorithm file is loaded into user RAM, the amount of user RAM available to the user is reduced by 1 MB. When the RAM device selection switch is turned on after the programmer is booted, the algorithm file is not loaded into user RAM until a device is selected.

When a device is selected, if the algorithm file was not loaded into this special user RAM area, it will be loaded before the device algorithm is selected. This causes the device selection time to be longer than usual for the first device after the switch is turned on. Subsequent device selection operations will take approximately one second, because no more disk accesses are necessary.

*Note: PPI Base information file (**adapters.sys**) is not loaded into RAM even if the RAM device selection switch is on. PPI Base information must be accessed from the disk when devices supported on the PPI Base are selected.*

ASCII	Description
***n*52]**	**Select Algorithm Source** Depending on the argument supplied, the ***n*52]** command sets the source for device programming algorithms to either the floppy disk or the Mass Storage Module. Valid arguments are listed and described below: 0 Select algorithms from the floppy drive 1 Select algorithms from the Mass Storage Module (not available on the 2900/3900)
***xxx*...*xxxx*53]**	**Save RAM Data to a Disk File** Saves the data in user RAM (in the range specified by the ***hhhhhh*:** and ***hhhhhh*;** commands) in a file. The filename is specified by the string preceding the 53].
54]	**Upload Device Footnote** (not available on AutoSite or the ProMaster 2500) Results in one of the following actions: If a selected device has a footnote and **devfnote.sys** is present, the footnote is uploaded. If devfnote.sys is not present, the footnote index list is uploaded. If a selected device has no footnote, error code B6 is returned.

Definition	**Number of Bytes**
Length of all footnotes to be uploaded in decimal bytes	2
<CR><LF>	2

Next is data for EACH footnote, organized as follows:

Text of footnote	**Variable**
<CR><LF>	2

If no device is selected, error code 30 is returned.

55] **Upload Device-specific Message** (not available on AutoSite or 2500) Uploads algorithm dynamic messages and device-specific failure or warning messages. The output format is as follows:

Definition	**Number of Bytes**
Length of message in decimal bytes	2
<CR><LF>	2

Next is data for the message, organized as follows:

Text of message	Variable
<CR><LF>	2

ASCII **Description**

56] and *nn*56] **Upload Memory Failure** (not available on AutoSite or PM2500)
The output format for 16-bit devices is as follows:

DDPAAAAAAAAHHHHhhhh

dd	Device socket
P	Verify pass indicator: H = High, N = Nominal, and L = Low V_{CC} Verify
AAAAAAAA	Failed memory location
HHHH	Data in User RAM
hhhh	Data read from the device

The output format for 8-bit devices is as follows:

ddPAAAAAAAAHHhh

HH	Data in User RAM
hh	Data read from the device

This command applies only to memory devices and uploads only the first failed location.

Use the 56] command for single device operations. The *nn*56] (where *nn* is the number of the device socket and is an integer number from 01 to 08) command is only available on UniSite.

57] and *nn*57] **Get Checksum of Operation** 57] returns the checksum of the last successful program, load, or verify operation. If the last program, load, or verify operation was not successful, the error code BC is returned.

On UniSite, the 57] command can be used for single device operations, and the nn57] is used with set operations (nn is the device number that ranges from 01 to 08).

58] **Uploads the System ID to the CRC Port** Output is in the following format:

HHHH HHHH HHHH

where each *H* is an ASCII hex character. This number is also displayed on the Update screen.

***n*59]** **Enable/Disable Capacitor Configuration Test** Enables or disables the capacitor configuration test. Valid arguments for *n* are:

0	Disable capacitor configuration test
1	Enable capacitor configuration test

5A] Returns a list of parameters in the form:

***dd*:*hhhhhhhh*:*hhhhhhhh*:*hhhhhhhh*:*hhhhhhhh*:*dd*:*hhhhhhhh*:*dd*>**

where *dd* is a two-digit decimal number and *hhhhhhhh* is an eight-digit hex number.

The parameters are in the following order:

I/O format number
Memory begin address
User data size
Device begin address
Device block size
Device width
I/O offset address
Data word width

ASCII	Description
5B]	Perform a clear vector operation that clears the vectors in RAM. If you encounter incompatible vector data, there may be vectors in RAM from the previously selected device or file. Perform a clear vector operation to correct this problem.
5C]	Load Custom Menu system files from source algorithm drive (derived by the algorithm media, *n*52] command, and the CM algorithm drive, *n*5F] command) to the RAM file buffer.
5D]	Transfer Custom Menu system files from RAM file buffer to the drive specified by the *n*5F] command. This command is used when creating a Custom Menu. For example, to create a Custom Menu, do the following: 1. Specify the ***n*4D]** command to select algorithm type where *n* can be 0 (default), 1 (extended), or 2 (Keep Current). (*n* = 3, Custom Menu, is **not** allowed.) 2. Specify the ***n*52]** command to select algorithm media where *n* can be 0 (select algorithms from floppy disk) or 1 (select algorithms from MSM). 3. Specify the ***n*5F]** command to select the CM algorithm drive where *n* can be 0 (place CM files on the disk in drive A), 1 (place CM files on the disk in drive B), or 2 (place CM files on drive I on an MSM). The source algorithm drive will be derived from the CM algorithm drive and the algorithm media. 4. Specify the **5C]** command to load CM system files from source algorithm drive to RAM file buffer. 5. Specify the **5D]** command to transfer Custom Menu system files from RAM file buffer to the drive specified by the *n*5F] command. 6. Specify the ***xxxxxxxx*33]** command to select the manufacturer name for the device you wish to place in your Custom Menu. 7. Specify the ***xxxxxxxx*64]** command to select the device part number for device you wish to place in your Custom Menu and load the device algorithm into the RAM file buffer. 8. Repeat the *xxxxxxx*33] and *xxxxxxxx*64] commands as necessary to select devices for your Custom Menu. 9. Specify the **5E]** command to transfer the CM algorithm files from the RAM file buffer to the drive specified by the *n*5F] command. Your Custom Menu can be used as a source for selecting devices. If you later wish to add a device to your Custom Menu, follow the same steps as described above, except **do not** use the 5C] and 5D] commands.
5E]	Transfer the CM algorithm files from the RAM file buffer to the drive specified by the *n*5F] command. See the **5D]** command for an example of how to use this command to create or add to a Custom Menu.
***n*5F]**	Select the CM algorithm drive, where *n* can be: 0 (Place CM files on the disk in drive A) 1 (Place CM files on the disk in drive B, if available) 2 (Place CM files on drive I on an MSM, if available) The source algorithm drive is derived from the CM algorithm drive and the algorithm media. If the algorithm media is MSM (152]), the source algorithm drive is always I (MSM). If the algorithm media is floppy (052]) and the CM algorithm drive is A (15F]), the algorithm source drive is B. Otherwise, the algorithm source drive is A. See the **5D]** command for an example of how to use this command.

ASCII	Description
60]	Get the number of sectors in the selected device (only if sector configuration is supported on the selected device). A two-digit decimal number is returned. If the device does not support sectors, this command returns zero.
***n*61]**	Get sector configuration settings (only if sector configuration is supported on the selected device), where *n* can be: 0 (get Erase settings for sectors) 1 (get Program settings for sectors) 2 (get Protect settings for sectors) *n*61] returns a hex value with each bit representing the data used to configure the corresponding sector. The LSB (least significant bit) corresponds to Sector 0. For example, if 344D were returned by 161] (get sector configuration settings for Program operation), the number has a binary representation of 0011 0100 0100 1101. The number 0011 0100 0100 1101 is read from right to left (sector 0 is on the far right) with a **0** meaning that the program operation is disabled for the sector and a **1** meaning that the program operation is enabled for the sector. The number 0011 0100 0100 1101 means that sectors 0, 2, 3, 6, 10, 12, and 13 can be programmed; the other sectors have programming disabled.
***nhhhhhhhh*62]**	Set sector configuration settings, where *n* can be: 0 (set Erase settings for sectors) 1 (set Program settings for sectors) 2 (set Protect settings for sectors) *hhhhhhhh* is a hex value with each bit representing the data used to configure the corresponding sector. The LSB corresponds to Sector 0. It is not necessary to fill in with leading zeroes. For example, to set sectors 0, 6, and 7 so that they would be programmed by the next programming operation, specify 1A162]. (The first 1 tells the programmer that you are changing the program settings for the sectors. The A1 is a hex value with a binary representation of 1010 0001, which tells the programmer to enable programming for sectors 0, 6, and 7. The 62] is the CRC code for the command.)
63]	Reboot the programmer.
***xxxxxxxx*64]**	Select the device part number for a device you wish to place in a Custom Menu (CM) and load the CM device algorithm into the RAM file buffer. See the **5D]** command for an example of how to use this command.
***A*65]**	Return the software version number (2900/3900 only), where A is the name of the programmer drive that contains the Boot Files Disk. The drive name is not case sensitive. The version number is displayed as a string of five characters, such as 3.80*x*, where *x* may be a blank.
6C]	**MSM Test** – Tests for the presence of the MSM (Mass Storage Module) within the programmer. If the MSM is mounted in the programmer, this returns a prompt. If the MSM is not present, an error 0xCC is generated.

ASCII	Description
n6D]	**Set Checksum Type** – Sets the parts of the device image in memory that will be included in the checksum returned by the "S" CRC command. This command is useful for devices that have information in memory beyond the main array. Sectored devices, for instance, that have protect information in RAM after the main array. This command can be used to control whether or not the protect information is included in the checksum. The single decimal digit that precedes the command can be one of the following: 0 Default. Whether the protect information is included in the checksum is determined by the algorithm. 1 Never. The protect information is not included in the checksum, whether the algorithm specifies that it should be or not. 2 Always. The protect information is always included in the checksum, whether the algorithm specifies that it should be or not. 3 Disable. This option specifies that the checksum operation should not be performed at all. In this case, the "S" CRC command will always return 0. Be careful when using this option!
n70]	**Get/Test Device Electronic ID –** Executes various functions concerning the Electronic ID of the selected device. The function executed depends on the value of the argument n, as shown below: 0 Verify the Electronic ID. The Electronic ID is read from the socketed device and compared with the ID expected by the current device algorithm. If the ID verifies, the command prompt is returned. If not, an error A2 will be generated. 1 Read the Electronic ID from the device. The ID is read from the socketed device and returned with the command prompt. 2 The Electronic ID expected by the current algorithm is returned with the command prompt. *Note: Reading the Electronic ID from a device requires that an algorithm is first selected. Even if the algorithm selected does not strictly match the device, the selected device algorithm must be similar to the socketed device or the Electronic ID cannot be properly read from it. If the selected device algorithm does not have an Electronic ID, an error A1 will be generated.* *The socketed device must be one that allows an Electronic ID to be read from it. Otherwise, damage to the device could occur.*
71]	**Stand-alone Device Erase –** Causes the socketed erasable device to be erased. This command performs the erase operation only and does not perform a byte by byte test to verify that the erase completed flawlessly.

ASCII	Description
hh…h72]	**Save System File –** Writes binary information contained in user RAM to a file on the system drive (drive H:). The hexadecimal number preceding the 72] determines how many bytes are written to the file. The name of the file is set by the 30] command. This command is only to be used for updating the programmer's system software. Use of this command for any other purpose may give unexpected results. When updating the system software, always use the files from the supplied CD. The files supplied on the floppy disks will not work properly with this instruction. Included with the system files on the CD in the directory corresponding to the programmer, there will be a file called "list72". This text file contains a list of all the system files that need to be transferred to the programmer's system drive, in the order that they must be transferred. The file contains one file name per line. This order is important because transferring the system.sys file causes all files on the system drive (drive H:) and the algorithm files from drive I: to be erased before the file is written.

Note: When performing a system software update to your programmer, the system files must be updated using 72], followed by the algorithm files using 73]. Updating the algorithm files first will not work because transferring system.sys will erase all the algorithm files.

Also note that although transferring system.sys with the 72] command will cause all the standard algorithm files to be erased from the I: drive, this will not erase any of the keep current or extended algorithm files. These files need to be individually cleaned from the drive when they are no longer needed.

ASCII	Description
hh…h73]	**Save Algorithm File –** Writes binary information contained in user RAM to a file on the algorithm drive (drive I:). The hexadecimal number preceding the 73] determines how many bytes are written to the file. The name of the file is set by the 30] command. This command may be used for updating the system software, but also may be used for uploading keep current and extended algorithms. If keeping your extended and keep current algorithms on this drive, be sure to periodically erase algorithms that are no longer in use to keep the drive from becoming full. When updating the system software, always use the files from the supplied CD. The files supplied on the floppy disks will not work properly with this instruction. Included with the system files on the CD in the directory corresponding to the programmer, there will be a file called "list73". This text file contains a list of all the algorithm related files that need to be transferred to the programmer's algorithm drive – in the order that they must be transferred. The file contains one file name per line.

Note: When performing a system software update to your programmer, the system files must be updated using 72], followed by the algorithm files using 73]. Updating the algorithm files first will not work because transferring system.sys will erase all the algorithm files.

Also note that although transferring system.sys with the 72] command will cause all the standard algorithm files to be erased from the I: drive, this will not erase any of the keep current or extended algorithm files. These files need to be individually cleaned from the drive when they are no longer needed.

77]	**Switch to Parallel Port –** Causes control of the programmer to be transferred to the parallel port. Once this command is executed, commands coming in through the Remote port will be ignored.

Note: Executing this command will cause control to transfer to the parallel port – whether you are connected to the parallel port or not. Also, care should be taken to execute this command only *if the programmer is an* xpi *programmer with a parallel port. In some versions of software, using this command with a programmer that does not have a parallel port will cause the programmer to* **lock up** *– forcing you to recycle power to regain control of the programmer. Normally, this command will not be needed because if the parallel port is connected and the Remote port is not, the programmer will automatically transfer control to the parallel port. Only if both the Remote port and the parallel port are connected when entering remote mode is this function useful.*

Also note that there is no command for transferring control back to the serial port. Once control has been transferred to the parallel port, the only way to use the serial port again is to transfer out of remote mode.

ASCII	Description
xx…x80]	**Echo Test –** Causes all command line parameters to be echoed back, followed by the prompt. This is intended as a test of the parallel port connection. The argument can consist of any binary characters of eight bits, up to 72 characters. This command can be used through the serial port also – but its usefulness is questionable. Also note that if this command is used through the serial port, the most significant bit of any argument characters will be set to zero before they are returned.
A7]	**Swap Bytes** Swaps the high bytes and the low bytes in a given memory range. Use the *hhhhhh*< and *hhhhhh*; commands to specify the memory begin address and the size of the memory range to swap. The memory begin address (specified by the *hhhhhhh*< command) added to the block size (specified by the *hhhhhh***;** command) cannot exceed the size of user memory. Also, the block size must be an even number. Entering a block size of 0 swaps all memory beginning with the address specified by the *hhhhhhh*< command. This command will not work if a logic device is selected. To swap high- and low-order nibbles, see the **Q** command for details.
DC]	**Device Check** Checks for the presence of a device in the socket. An empty causes the **DC]** command to return an F. Error code 3B is returned after the **X** command is sent. If a device is in the socket, further device checks are done if the device supports insertion and socketing tests. For example, the **DC]** command returns a device insertion error if the device is mis-socketed. If a device is in the socket and no continuity errors occur, the **DC]** command returns a normal > prompt.
DF] (UniSite w/ SetSite)	**View Status of Sockets** Returns the results of the previous device operation for all eight sockets (SetSite). The results are returned in groups of eight 2-digit hex numbers, separated by a space. Each group represents data for a particular device. The first two hex characters contain data for the device in socket one, etc. The results are cleared after receiving the **&** command. Each bit contains different device status information: **Bit** **Status if set to one** 7 Error was detected 6 Non-blank device error 5 Device testing or overcurrent error 4 Invalid electronic ID error 3 Illegal-bit error 2 Programming error 1 Verify error 0 Device is in the socket
EB]	**Input JEDEC Data From Host** Sets the I/O Format to 91 and waits for JEDEC-formatted data to be sent to the programmer through the Remote port. This command is valid only for logic devices. All the fuse map and structured vectors will be received and placed in RAM at the corresponding position.
EC]	**Output JEDEC Data To Host** Sets the I/O Format to 91 and sends JEDEC-formatted data from the programmer through the Remote port. This command is valid only for logic devices. The fuse and vector data in RAM must be valid for the selected logic device. The complete fuse map and structured vectors are output.

ASCII	Description
FC]	**Restore CRC Entry Default Parameters** Sets the CRC parameters to the original factory defaults.
FD]	**Restore User-defined CRC Parameters** Sets the CRC parameters to the last saved user-defined CRC parameters. The user-defined CRC parameters are the same as the CRC factory defaults if no user-defined parameters have been changed.
FE]	**Save User-defined CRC Parameters** Saves the current CRC parameters to a disk as the user-defined CRC parameters. For UniSite, make sure the System disk is not write protected and is in a drive. For other programmers, make sure the Boot Files disk in not write protected and is in the drive.

Error Status Word

The **F** command returns an 8-character, 32-bit error status word. The following table shows the format of the error status word.

The 8-character word is broken into four 2-character groups. The first 2-character group defines receive errors, the second group defines programming errors, the third group defines I/O errors, and the last group is unused. Each 2-character group contains eight bits, with each bit representing an error or error type. For each bit, a 1 represents an error and a 0 represents no error.

	Bit No.	Value	Description
Receive Errors	31	8	ANY ERROR. If the word contains any errors, the most significant bit (bit 31) will be high.
	30	4	Not used
	29	2	Not used
	28	1	Not used
	27	8	Not used
	26	4	Serial-overrun error (42)
	25	2	Serial-framing error (41,43)
	24	1	Not used
Programming Errors	23	8	Any device-related errors
	22	4	Not used
	21	2	Not used
	20	1	Not used
	19	8	Device not blank (20)
	18	4	Illegal bit (21)
	17	2	Nonverify (23, 24, 2B, 2C, A2)
	16	1	Incomplete programming (22, 2A, 30, 31)
I/O Errors	15	8	I/O error (46)
	14	4	Not used
	13	2	Not used
	12	1	Compare error (52)
	11	8	Sumcheck error (82)
	10	4	Record-type error (94)
	9	2	Address error (27)
	8	1	I/O Format error (84, 90)
Unused	7	8	Not used
	6	4	Not used
	5	2	Not used
	4	1	Not used
	3	8	Not used
	2	4	Not used
	1	2	Not used
	0	1	Not used

Example:

What errors are indicated in this error status word: 80888000 ?

8	the word contains error information
0	no receive errors
8	device related error
8	device is not blank (error 20)
8	I/O error
0	no errors
0	no errors
0	no errors

Note: *The numbers in parentheses are the programmer error codes, defined in the error codes section of this chapter.*

An error can cause as many as 3 bits to be high: the bit which represents the error, the most significant bit of the 8-bit word in which the error bit occurs, and the bit 31.

After being read, the error-status word resets to zero.

CRC Error Codes

Following is a list of error codes that appear while the programmer is being operated in the computer remote control mode. These error codes will be returned by the programmer after it receives the **X** command. Normally, you should send the **X** command after the programmer sends an **F** in response to a command. The list is in numerical order, according to the error code.

Error	Explanation
1F	**Cannot erase device error** Appears after the programmer was not able to erase an EEPROM. The device may be defective; try another device.
20	**Non-blank device** Appears after the programmer has performed a blank check on a device and has detected bits that are not in their erased or blank state and are not illegal bits. This error is the result of either the **B** (Blank Check) command or a **P** (Program) command with the blank check option set previously by the extended command **2A** (Enable Programming Option).
21	**Illegal bit error** Appears when the programmer has detected a device that has a bit programmed to the incorrect state. When this error code appears, try erasing the part (if possible) and then attempt to program the part again. (This error indicates a fatal condition in non-erasable devices.) If this error code continues to appear, it may be because the device is defective. Discard the part and try another device. The illegal-bit check error occurs as the result of either a **T** (Illegal Bit Check) command or a **P** (Program) command with the illegal bit option set previously by the extended command ***hhh*2A]** (Enable Programming Option).
22	**Device programming error** Appears when the programmer detects a defective memory cell in a device during programming. If this error code appears, try another device.

Note: *The two following errors have the same error code. For either error to appear, you must have selected command* **n***23] (Select Verify Option). If* ***1*** *was specified as the variable, use the first description. If* ***2*** *was specified, use the second description.*

Error	Explanation
23	**Verify data error (V_{CC}** Nominal) Appears when the programmer has performed a Verify and has found a memory cell that was not programmed correctly. The device was verified while being operated with its normal operating voltage applied. When this error code appears, try another device.
23	**Verify data error (V_{CC}** low) The programmer performed a Verify operation and found a memory cell that was programmed incorrectly. The device was verified while being operated with its lowest operating voltage applied. When this error code appears, attempt to program the device again. If this error code reappears, try a different device.
24	**Verify data error (V_{CC}** high) The programmer performed a Verify operation and found a memory cell that was programmed incorrectly. The device was verified while being operated with its high operating voltage applied. When this error code appears, attempt to program the device again. If this error code reappears, try a different device.
27	**End of user RAM exceeded** There is not enough user RAM for the amount of data you want to load into it or program from it. You may have the device block size set too large, or the beginning RAM address too high. The operation can still be performed, but only part of the device will be programmed.

Error	Explanation
28	**Fatal device-specific programming error** Generic error code generated by the device-specific algorithm. Correct the error condition before trying further commands.
29	**Non-fatal device-specific programming error** Generic error code generated by the device-specific algorithm. This error is non-fatal and is for information purposes only. In most cases, the operation was performed successfully.
2A	**Device Insertion error** Appears when the device socket is not in the locked position, the device is inserted backwards or is not bottom-justified in the socket, or the device pins are not making good contact. Check the device's continuity in the socket, then retry the operation. If the same error code appears, try a different device.
	Note: The two following errors have the same error code. For either error to appear, you must previously have selected command **n23]**, *where n = 0, 1, or 2. If a 0 is chosen, there will be no error condition. (Select Verify Option.) If* ***1*** *was specified as the variable, use the first description. If* ***2*** *was specified, use the second description.*
2B	**Structured test error (V_{CC}** Nominal) The programmer performed a functional test on a logic device and detected a failure. If you selected Verify Passes = 1, the programmer tried to verify the logic device at its normal operating voltage. Try another device.
2B	**Structured test error (V_{CC}** low) The programmer performed a functional test on a logic device at the low voltage and detected a failure. If you selected Verify Passes = **2**, one pass is performed while the lowest specified operating voltage is applied to the device and the second pass is performed while the highest specified operating voltage is applied to the device. Try another device.
2C	**Structured test error (V_{CC}** high) The programmer performed a functional test of a logic device at the high voltage and detected a failure. If you selected Verify Passes = **2**, one pass was performed while the lowest specified operating voltage was applied to the device, and the second pass was performed while the highest specified operating voltage was applied to the device. Try another device.
2D	**Base/Adapter for device not installed** The device you selected cannot be programmed in the base that is presently installed because the device is not supported by the installed base. If you select a device that uses the PPI base, make sure the correct adapter is installed. Install the correct base/adapter and try again.
2E	**Programming hardware hasn't passed self-test** Occurs when a programming operation is attempted and the self-test had previously failed for critical hardware. Return to local mode and check the self-test screen.
2F	**Insufficient pin driver boards installed for the device selected** The device you are trying to load, program, verify, or check needs more pin driver boards than your programmer has.
30	**Bad algorithm** The selected algorithm is somehow corrupted and cannot be used. Contact Data I/O.
31	**Device over-current fault** You attempted to program a socketed device whose programming current is higher than the device you selected on-screen. The device may be faulty. Insert another device into the socket and try the operation again.
3B	**No device present** No device was in the socket when a device operation was tried.
40	**I/O initialization error** Appears after an attempt to initialize the Remote port has failed. Check connections and attempt the operation again.

Error	Explanation
41	**Serial-framing error** The Remote port detected a start bit, but the stop bit is positioned incorrectly. Check the baud rate and stop bit setting for the Remote port, or use hardware handshaking.
42	**Serial-overrun error** The programmer was unable to service the characters received by the Remote port. Check the baud rate and stop bit settings for the Remote port, or use hardware handshaking.
43	**Serial framing/overrun error** This is a combination of serial-framing error 41 and overrun error 42. Check the baud rate and stop bit settings for the Remote port, or use hardware handshaking.
46	**I/O timeout** Too much time passed before the programmer received a data file during a download operation. Use Select I/O Timeout Command (=) to change I/O timeout period.
52	**Data verify error** The data from the Remote port did not match the data in RAM. Check the data and try the operation again.
75	**Security Fuse Violation** You tried to load, program, or verify data from a device whose security fuse is programmed. Use a master device whose security fuse is still intact.
77	**Security fuse programming error** The programmer cannot program the security fuse. The device you are trying to program may be defective. Try programming another device.
79	**Preload not supported by this device** A preload vector in the programming data cannot be applied to the logic device.
81	**Serial-parity error** The Remote port detected incoming data that had incorrect parity. Check the parity setting for the Remote port.
82	**Sumcheck error** The sumcheck of the data received (as the result of a download) did not match the sumcheck downloaded from the host computer. The host computer sends a transmission sumcheck, and possibly a fuse map sumcheck, as a part of the data record. The programmer compares those sumchecks with the sumchecks it created on that same data. If the two sumchecks do not match, this error code will appear, indicating that some of the data transmitted by the host was not received by the programmer. Try the operation again. If the problem continues, verify that the sumchecks generated from the host are correct. If the sumchecks are correct, contact Customer Support.
84	**I/O format error** There is a compatibility problem with the data translation format you are using. Check the format of the data. The Translation Formats section in your **Programmer's User Manual** describes all the data translation formats supported by the programmer. Or try sending a different translator format. If format 04 is selected, this error can indicate an illegal parameter error. Since this format is word oriented, set the following parameters to even values: I/O offset, memory begin address, user data size, and upload record size.
88	**Invalid number of parameters** Appears when a CRC command is preceded by an invalid number of parameters. Correct the situation and re-issue the command.
89	**Illegal parameter or invalid operation selected** Appears if an illegal parameter or option precedes a CRC command. For example, this code is returned if you set the number of verify passes to 0 and then performing a verify. This message also appears if you attempt an operation that is not compatible with the selected device, such as if you attempt a byte swap with a logic device selected.
8A	**Operation not enabled.**

Error	Explanation
8B	**Error restoring/saving CRC user-defined parameters or restoring CRC entry defaults.** An error occurred while attempting to restore or save CRC user-defined parameters (commands **FD** or **FE**) or restoring CRC entry defaults (command **FC**). This error can also occur if a problem occurs when attempting to restore an algorithm file. For UniSite, make sure the System and Algorithm disks (if restoring an algorithm, use the disk that contains the algorithm) are not write-protected and are in the disk drives. For other programmers, make sure the System/Algorithm disk (if restoring an algorithm, use the disk that contains the algorithm) is not write-protected and is in the disk drive. Use System/Algorithm Disk 1 if restoring parameters or entry defaults.
8E	**File error** A disk file error occurred during a command that accesses a disk file, such as Load File from Disk, Yield Tally, or Select Device. If you are loading a file from disk, check the filename; it may not exist or may be misspelled. If the filename is spelled correctly, make sure that the disk in the programmer's disk drive contains the file you are trying to access, such as a Keep Current algorithm. If you are doing a Yield Tally or selecting a device, the disk is probably write-protected. Remove the write-protection and retry the operation. This error code also occurs when the Algorithm Save area is exceeded during a Save Configuration operation.
8F	**NON-JEDEC data present in RAM or disk file, or else a NON-logic device was selected with a JEDEC I/O translation format selected** For a Load File from Disk or device operation, check your JEDEC data. For an upload operation, select the logic device for the JEDEC file to be uploaded, or select a different I/O format for a memory device.
90	**Illegal I/O format** You tried to select an I/O format that is not supported by the programmer, or you attempted to select a non-JEDEC format when a logic device was selected. Select a valid format. See the Translation Formats section in your **Programmer's User Manual** for a list of supported formats.
94	**Data record error** The data that you attempted to transfer did not conform with the selected translation format; edit the data file so that it matches one of the programmer's supported translator formats. See the Translation Formats section in your **Programmer's User Manual** for output samples of each translator.
97	**Block move error** A block move within RAM has violated the RAM boundaries. Check the memory begin address and memory block size and try the operation again.
98	**End of device exceeded** There is not enough room in the device to hold all the data you have specified. The device beginning address may be set too high, the block size set too high, or you may need a larger device. Although the operation may still be performed, only part of the data will be programmed into the device.
99	**End of file exceeded** The memory block size and memory begin address parameters specified in the Programming screen are too large for the data file to be used for programming. Change the memory block size and memory begin address file size parameters so they are small enough to accommodate the data file. You can perform the operation without changing anything, but only part of the device will be programmed.
9A	**Algorithm disk cannot be found** Appears if you are selecting a device and do not have an Algorithm disk in a disk drive of your programmer. Insert the Algorithm disk in the drive and send the device selection command again.
9B	**Incompatible system/algorithm revision numbers.** This error code is returned during a device selection operation when the version number of the algorithm file is not compatible with the system software. Insert the disk with the correct algorithm file.

Error	Explanation
9C	**Invalid command for this mode** This error occurs if a command received by the programmer is valid only in set or gang programming mode and the programmer is running in single device mode. This error also occurs if the programmer has set mode enabled but the selected device is not supported in set or gang programming mode.
9D	**I/O address beyond range of data format selected** An I/O address exceeded the highest value allowed in the address field of the data format selected. Before it performs an upload or output to disk operation, the programmer calculates the highest I/O address that will be output based on the parameters you supply, and aborts the operation if the I/O address is too large for the data format selected. The formula to calculate the highest I/O address is: `Highest I/O address = I/O addr offset + User data size – 1` Either select a different data format (one that supports the I/O addresses for the transfer operation) or decrease the value of the I/O offset address and/or the User data size to achieve I/O addresses within the range of the data format selected. The I/O addr offset parameter is considered an unsigned value. If it is set to the special default value of FFFFFFFF, it is treated as a value of 0. Refer to the Translation Formats section in your **Programmer's User Manual** for a list of the highest I/O address allowed for each format.
9E	**Illegal value for Vcc voltage** The value entered for the Vcc variable voltage for the device was not a legal voltage.
9F	**Illegal variable voltage operation** Contact Data I/O.
A0	**System Files disk not found** The programmer needs the System Files disk present in order to access a file on that disk. Insert the System Files disk. If the programmer has more than one System Files disk, insert any of these disks.
A1	**No Electronic ID** The device does not contain an electronic ID. Turn off the Electronic ID option or change devices.
A2	**Electronic ID verify error** The device you tried to program did not have the correct electronic ID. Insert the correct device in the socket, or select a different device.
AA	**Variable verify not supported for selected device.**
AB	**Unable to load system file from system disk** You tried to exit or suspend CRC and the System or Boot Files disk was not in the disk drive. Make sure the disk is in the drive when you exit or suspend CRC.
AC	**Security violation** You tried to use a new version of system software that has not been installed. Exit remote mode and run the Update command to install the new version.
AE	**Keep Current algorithm disk not found. Insert your Keep Current algorithm disk** The Keep Current algorithm file for the specified device is not found. Insert the disk with the Keep Current algorithm file for the specified device, and try again. (Sometimes the 8E error can occur in place of the AE error.)
AF	**Operation not allowed because the device was selected by family/pinout code** This error code is returned when you try some device operations, such as Compare Electronic ID, after having used a family/pinout code (the CRC **@** command) to select a device. Use the ***xxx…xxxx*33]**, ***xxx…xxxx*34]**, and ***n*40]** commands to reselect the device using device manufacturer and device part number. To avoid this error, rewrite your CRC driver to use the device manufacturer and device part number to select devices.

Error	Explanation
B0	**Capacitor Configuration Error** (AutoSite and PM2500 error) Due to the configuration of the capacitors on the currently installed programming module, you will not be able to program the currently selected device. Normally, this error occurs when a capacitor on the programming module is located on a device pin that is not a GND, VCC, or NC. Programming the currently selected device with the currently installed programming module could result in a marginally programmed device. A marginally programmed device may test properly, but is prone to failing in circuit. To avoid this error, install a programming module with capacitors located only on GND or VCC pins. Contact Customer Support for more information on the programming module required to program the currently selected device.
B1	**Block not allowed for bulk erase** You tried to bulk erase part of a device and the device does not support partial bulk erasing. To bulk erase the device, you must erase the entire device by setting the Device Begin Address to 0 and the Memory Block Size to the size of the device.
B2	**Partial device operation not allowed** An attempt to program a device failed because of one or both of the following reasons: 1) the last programming operation attempted to program only a portion of the selected device (the whole device must be programmed when a programming operation is used on it), or 2) the User Data is incorrect for the selected device. To make sure the entire device is programmed, set both the **Device Begin Address** and the **Device Block Size** to **0** (when Device Block Size is set to 0, the system sets the Device Block Size to the size of the entire device(s)). To make sure the **User Data** is correct in the device operation, enter **0** in the **User Data Size** field (the system sets the User Data Size to cover the entire device(s)).
B4	**Odd Memory Begin Address or User Data Size Incompatible with Data Word Width** You tried a device operation on a 16-bit (or larger) device and either the Memory Begin Address is set to an odd number, or the User Data Size is not compatible with the Data Word Width selected. Frequently, this happens when a 16-bit device is used and the User Data Size (defined in bytes) is an odd byte count. Adjust your User Data Size (must be even number) or the Memory Begin Address.
B6	**Algorithm doesn't have a footnote** The programmer was unable to display a footnote (device information) of the currently selected device because it could not find the information on the disk that is currently installed in the disk drive. Insert the algorithm disk that contains the device information for the currently selected device. If the correct disk is inserted, then no device information is available for the device. For further device information, refer to the *Device List On a Disk.*
B8	**Algorithm Set 1 disk not found** A device select operation failed because the device's algorithm resides on the disk that contains algorithm set 1, and this disk is not installed in the programmer. Insert the disk that contains algorithm set 1.
B9	**Algorithm Set 2 disk not found** A device select operation failed because the device's algorithm resides on the disk that contains algorithm set 2, and this disk is not installed in the programmer. Insert the disk that contains algorithm set 2.
BA	**Yield Tally access/update error** There was a problem during yield tally-related operation (43], 46], 042A], 042B]). Yield tally operations require the presence of a write-enabled System disk (UniSite) or Boot Files disk (2900, 3900, AutoSite, ProMaster 2500).
BB	**Incomplete boot/algorithm file set.**

Error	Explanation
BC	**Last device operation checksum not available** No successful Load, Verify, or Program operation has been done since the programmer was last powered up, or the last Load, Verify, or Program operation was unsuccessful.
BD	**Algorithm Set 3 disk not found** A device select operation failed because the device's algorithm resides on the disk that contains algorithm set 3, and this disk is not installed in the programmer. Insert the disk that contains algorithm set 3.
BE	**Custom Menu disk not found** A device select operation from a Custom Menu file failed because the device's algorithm cannot be found on the disk currently installed in the disk drive. Insert the Custom Menu disk that contains the algorithm for the device.
BF	**User RAM files exist** A 5C] command was attempted but cannot be used because user RAM files are detected in user RAM. To proceed with the 5C] command, clear all RAM files using the 39] command.
C0	**Too many revisions of the same Custom Menu algorithm** A Custom Menu algorithm cannot be placed on a disk because Custom Menu algorithms for the **same device** and **same software version number** already exist with the **.CMA** to **.CMZ** extensions (26 CM algorithms total).
C1	**Invalid algorithm type** A Custom Menu was selected as the algorithm type for building a Custom Menu. Use the ***n*4D]** command to select a different algorithm type (supported algorithm types are default, extended, and Keep Current).
C2	**Custom Menu algorithm buffer full** All available user RAM has been used (the Custom Menu algorithm building process uses user RAM as a buffer). You can use the **5E]** command to save the contents of the RAM buffer onto the Custom Menu algorithm disk. If you choose not to do so, you must use the **39]** command to free up user RAM.
C3	**Extended algorithm disk not found** A device select operation from an extended algorithm file (14D] command) failed because the algorithm for the device cannot be found on the disk installed in the disk drive. Insert the disk containing the extended algorithm for the device.
C4	**Algorithm Set 4 disk not found** A device select operation failed because the algorithm for the device resides on the disk that contains algorithm set 4 and this disk is not installed in the programmer. Insert the disk that contains algorithm set 4.
C5	**Algorithm Set 5 disk not found** A device select operation failed because the algorithm for the device resides on the disk that contains algorithm set 5 and this disk is not installed in the programmer. Insert the disk that contains algorithm set 5.
C6	**Algorithm Set 6 disk not found** A device select operation failed because the algorithm for the device resides on the disk that contains algorithm set 6 and this disk is not installed in the programmer. Insert the disk that contains algorithm set 6.
C7	**Algorithm Set 7 disk not found** A device select operation failed because the algorithm for the device resides on the disk that contains algorithm set 7 and this disk is not installed in the programmer. Insert the disk that contains algorithm set 7.
C8	**Algorithm Set 8 disk not found** A device select operation failed because the algorithm for the device resides on the disk that contains algorithm set 8 and this disk is not installed in the programmer. Insert the disk that contains algorithm set 8.
CA	**Sector operations not supported for this device** The currently selected algorithm does not support sector operations. This is usually caused by executing the 61] or 62] commands with a device algorithm selected that does not support sector operations.

Error	Explanation
CB	**Sector protect operations not supported for this device** The currently selected algorithm does not support the protection of sectors. This is usually caused by executing the 261] or 262] commands with a device algorithm selected that does not support the protection of sectors.
CC	**No MSM operations supported for this programmer** The attempted operation requires the presence of an MSM (Mass Storage Module) and the programmer does not have an MSM installed.
D1	**RAM File Internal Error** Call Customer Support.
D2	**RAM file not found** An operation with a RAM file, such as program device from a RAM file (using a P command after a 231] command), failed because the RAM file (specified by the *xx...xxx*30] command) cannot be found in user RAM. Make sure your data source (specified by the *n*31] command) and file name are correct.
D3	**RAM File Internal Error** Call Customer Support.
D4	**RAM File area exhausted** An attempt to create a RAM file failed because there was not enough RAM space remaining for RAM files or there were not enough directory entries left for RAM files..
D5	**Port transfer error** Appears when you try to transfer data over a serial port which is not properly connected. If this error occurred after using the ***n*3C]** command to set the data port to the Terminal port, ensure that the Terminal port is properly connected.
E0	**Socket not empty during self-test** The self-test operation was invoked while a device was in the socket. The self-test cannot safely or accurately proceed with a device in the socket.
E1	**Floppy diskette not installed during self-test** The floppy diskette drive cannot be tested without the presence of a floppy diskette. Insert any formatted floppy and try the operation again.
E2	**Drive B is not available** A self-test of the programmer's drive B was attempted although the programmer is not equipped with a second floppy diskette drive.
E3	**No PSM or FSM installed** A self-test of the PSM and FSM was requested, but neither the PSM nor FSM are actually installed.
E4	**Self-test failure –** The requested test failed.
E5	**MSM format failure** A self-test of the MSM showed that the format information on the internal hard drive is not correct.
E6	**MSM test failure** A self-test of the MSM indicated a problem with either reading or writing information to the internal hard drive.
E7	**Not installed** The requested item to be tested was not installed.
E8	**Unknown adapter** The installed adapter cannot be identified.
E9	**Not calibrated** The Calibration self-test must pass before the PCU test can be performed.
EA	**PCU not tested** The Calibration and PCU self-tests must both pass before the pin driver board test can be performed.
FE	**Undefined error** An error occurred that the CRC program could not categorize. Document the method in which the error occurred and call Customer Support to report the problem.

Error	Explanation
FF	**Operation aborted at programmer** This error can occur if the current SetSite operation is halted prematurely: for example, if the programmer was in the process of programming devices and the SetSite socket lever is moved to the Open (stop) position. To correct the problem, close the socket lever and restart the operation.

XPI Parallel Port Interface

Introduction

This section covers technical information on the behavior of the *xpi* parallel port interface for those who wish to write their own programmer driver software. Data I/O *xpi* programmers can still be driven in Remote Mode through the serial port just like their predecessors. The parallel port functions in Remote mode in a manner very similar to the Remote serial port - only faster. Since command/prompt format of Computer Remote Control is the same whether you operate the programmer through the serial or the parallel interface, this section will only cover the differences between the two.

Overview

The *xpi* parallel port on the programmer is designed to be connected to a standard printer port on the IBM compatible personal computer. The original printer port was designed to pass information in only one direction: from the computer to the printer. Therefore, in order to get information to flow in both directions through this port, signals on the printer port will need to be used in ways they were never originally intended. Status lines will be employed to send information back to the computer from the programmer and one of the control lines will be commandeered to indicate the direction of data flow.

Sending a command to the programmer is fairly straightforward. The computer has complete control over the direction of information transfer between the computer and the programmer through the **Direction** line. The computer sets the **Direction** line to the “transmit” state indicating a transfer from the computer to the programmer. It then waits for a ready status from the programmer on the **-Ready/Nybble** line. At that time, it presents data to the programmer and toggles the **-Strobe** line. The computer then waits for the ready status again, then either sends another character or sets the **Direction** line back to the “receive” state, indicating to the programmer that the information transfer is complete.

Normally, the **Direction** line will spend most of its time in the “receive” state. It will only be pulled to the “transmit” state when the computer wishes to send a new Remote Mode command, or wishes to discontinue the operation in progress. The programmer is only able to send information to the computer when the **Direction** line is in its “receive” state.

Receiving information from the programmer is a bit trickier than sending it. Since the printer status lines must be used to send information to the computer, and there are very few of these status lines, information can only be sent to the computer four bits at a time. Four bits of information is often referred to as a “nybble”. When the **–Ready/Nybble** line goes low, this indicates that the low order nybble of information can be read from the printer status lines. Once the computer has read the low order nybble it signals that it is ready for the next nybble by toggling the **–Strobe** line. When the **–Ready/Nybble** line goes back high, it indicates that the high order nybble of information can be read from the printer status lines. Once the computer has read the high order nybble, it signals this by toggling the **–Strobe** line again. This continues until all bytes of the transfer are complete.

Although the computer can communicate the start and end of the information it wishes to send to the programmer by changing the state of the **Direction** line, there is no analogous line that the programmer can use to indicate the start and end of the information it sends back. The computer must infer when it has received the last byte from the programmer by interpreting the information itself.

Signal Descriptions

The following is a table that shows the pin numbers of the computer's printer port, the name of the signal on each pin as it applies to a standard printer, and the name of the signal on the same pin as it applies to the Data I/O *xpi* parallel port.

	Standard Parallel Port Function Name	**Data I/O Parallel Port Function Name**
Pin 1	-STROBE	**Direction**
Pin 2	DATA 0	**+TD0**
Pin 3	DATA 1	**+TD1**
Pin 4	DATA 2	**+TD2**
Pin 5	DATA 3	**+TD3**
Pin 6	DATA 4	**+TD4**
Pin 7	DATA 5	**+TD5**
Pin 8	DATA 6	**+TD6**
Pin 9	DATA 7	**+TD7**
Pin 10	-ACK	**-RD1/5**
Pin 11	+BUSY	**-RD0/4**
Pin 12	+PE	**-RD2/6**
Pin 13	+SLCT	**-RD3/7**
Pin 14	-AUTO FD XT	N.C.
Pin 15	-ERROR	**-Ready/Nybble**
Pin 16	-INIT	**-Strobe**
Pin 17	-SLCT IN	N.C.
Pin 18	GND	GND
Pin 19	GND	GND
Pin 20	GND	GND
Pin 21	GND	GND
Pin 22	GND	GND
Pin 23	GND	GND
Pin 24	GND	GND
Pin 25	GND	GND

Note: *The signal descriptions will refer to "bringing (some signal) high" or "bringing (some signal) low". This refers to the actual voltage on the pin of the computer's printer port. The same is true for all waveform diagrams. All signals are five-volt TTL levels. A "high" refers to bringing the signal above 3.5 volts and a "low" refers to bringing the signal below 0.8 volts. Each of these signals is controlled by bits in hardware registers inside the computer. Please note that it may be necessary to set one of these register bits low in order to bring the signal at the port pin high – and vice-versa. Again, to avoid confusion, this document will* *refer* only *to the TTL level present on the actual pin of the computer's parallel port and will make no reference to the values present in the computer's internal registers.*

Following is a description of each of the signals listed above:

Direction (pin 1)

This pin sets the direction of data transfer on the parallel port and is controlled by the computer. It is pulled high internal to the programmer with a 5mA current source. When this line is high, data may flow from the programmer to the computer. When this line is low, information (usually commands) may flow from the computer to the programmer.

There are only two restrictions on when this line may change. One is that when transferring information from the computer to the programmer, you must wait until the **–Ready/Nybble** line goes low before bringing the **Direction** line high or the last byte sent to the programmer may be lost. The other restriction is that you must wait at least 7 microseconds after the last high to low transition of the **–Strobe** line before bringing the **Direction** line low.

+TD0 through +TD7 (pins 2 through 9)

These are the eight data lines that transfer information from the computer to the programmer. They are controlled only by the computer. These lines have positive logic. That is, a high voltage on the pin will transfer a logic 1 to the programmer. **+TD0** is the low order bit and **+TD7** is the high order bit.

-RD0/4, -RD1/5, -RD2/6, -RD3/7 (pins 11, 10, 12, 13, respectively)

These are the four data lines that transfer information from the programmer to the computer. They are controlled only by the programmer. These lines have negative logic. That is, a high voltage on the pin will refer to a logic 0 from the programmer. The low order nybble is always transferred first on these pins. That is, data bits 0 through 3. Then the high order nybble will be transferred, or data bits 4 through 7.

-Ready/Nybble (pin 15)

The function of this pin is different depending on the state of the **Direction** line. If the **Direction** line is high, and data is being transferred from the programmer to the computer, this line functions as the **Nybble** line. If the **Direction** line is low, and data is being transferred from the computer to the programmer, this line functions as the **–Ready** line.

When transferring data and commands from the computer to the programmer, this line goes low when the programmer is able to receive data. This line will go high a few nanoseconds after the falling edge of the **–Strobe** line. It will go low again after the programmer has read the states of the **+TD0** through **+TD7** lines. Once this line has gone low, the next data may be set up on the **+TD0** through **+TD7** lines or the **Direction** line may be set high to indicate the end of the data or command.

When transferring data or response information from the programmer to the computer, this line goes low when the low order four bits of data are present on the **–RD0/4** through **–RD3/7** lines. This line and the data will change at the same time. After the computer has read this information, it must then toggle the **-Strobe** line. A few microseconds after that, this line will go high indicating the presence of the high order four bits of data on the **–RD0/4** through **–RD3/7** lines. Again, this line and the data will change at the same time. Again, the **–Strobe** line should be toggled after this information is read. If there is more data, in a few microseconds, the **Nybble** line will go low again.

-Strobe (pin 16)

This line is normally held high. Its function is triggered by its high to low transition. It therefore only needs to be held low for a few nanoseconds at a time. This pin is pulled high internal to the programmer with a 5mA current source.

The function of the **–Strobe** pin is different depending on the state of the **Direction** line. If the **Direction** line is high, and data is being transferred from the programmer to the computer, this line indicates to the programmer that the computer has received the last nybble sent by the programmer and that it may send the next. If the **Direction** line is low, and data or commands are being transferred from the computer to the programmer, this line indicates that data is present on the **+TD0** through **+TD7** lines that the programmer needs to read.

Basic Operations

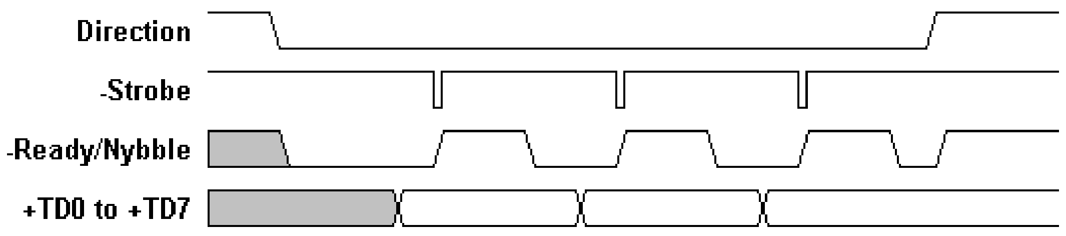
This section describes how to do various operations with the parallel port. The most obvious operation is the transfer of data and commands. There are other functions, however, that are performed in a manner very different from that of the serial ports. Discontinuing a data transfer, for instance, or just powering on the programmer, may be handled in unexpected ways when the parallel port is involved. That's what this section is about.

Sending commands to the programmer

The programmer is at your beck and call. There are no maximum timings and no timeouts on the parallel port. The computer has full control and may run the parallel port as slowly as it wants or as quickly as the programmer is able to take it.

Figure 3 below shows the rough appearance of a typical command sent from the computer to the programmer. In this case, it is a three-byte command.

Figure 3. Typical waveform of data transfer from the computer to the programmer.



The last byte of a Remote mode command to the programmer must be a carriage return. It is also possible to end a command with a carriage return and a line feed. The line feed may precede or follow the carriage return or be absent altogether, but the carriage return character must be present. Note that all line feed characters are ignored, but NULL characters (ASCII 0x00) are not. The serial port ignores NULL characters, but a NULL character sent to the parallel port is significant.

*Note: The **Direction** line signals the start and end of a Remote mode command. The programmer, in fact, will do nothing but wait for more characters while the **Direction** line is held low. It will wait for the **Direction** line to go high before processing the command. Note also that exactly one command must be sent between transitions of the **Direction** line. When sending the escape character, it is considered a complete command in itself.*

To send a command to the programmer, first bring the **Direction** line low. Within a few nanoseconds, the **–Ready/Nybble** line will go low, indicating that the programmer is ready to receive the first character of the command.

Immediately after the **–Ready/Nybble** line goes low, data for the character may be presented to the programmer on the **+TD0** to **+TD7** lines. After that, a few nanoseconds must be allowed for the data to propagate to the programmer's data register, then the **–Strobe** line may be brought low. The high to low edge of the **–Strobe** line both alerts the programmer that data is available to be read and causes the **–Ready/Nybble** line to go high. It will take a few nanoseconds for the high to low transition of the **–Strobe** line to propagate through to the **–Ready/Nybble** line. Since only the high to low transition of the **–Strobe** line is important, the **–Strobe** line may be brought high again only a few nanoseconds after bringing it low.

Although Figure 3 shows the computer changing the data on the **+TD0** through **+TD7** lines after the **–Ready/Nybble** line has gone low, the data may actually be changed a few nanoseconds after the high to low transition of the **–Strobe** line. Before the **–Strobe** line may be brought low again to send this new byte of data to the programmer, though, the computer must wait for the **–Ready/Nybble** line to go low.

The programmer will generally take a few microseconds to process the data between the falling edge of the **–Strobe** line and the falling edge of the –**Ready/Nybble** line. The next **–Strobe** line high to low transition may occur immediately after the **–Ready/Nybble** line has gone low.

Note: *When the last byte of the command (usually a carriage return) has been sent to the programmer by toggling the –Strobe line, the computer* must *wait for the –Ready/Nybble line to go low again before bringing the Direction line high, signaling the end of the command. Failure to wait for the –Ready/Nybble line to go low before bringing the Direction line high may result in the last character of the command being dropped.*

Sending data to the programmer using the "I" command

There is a special protocol for sending data to the programmer with the Remote mode "I" command. Since the "I" command is a command in itself, the **–Direction** line must be brought high after sending the "I" followed by its carriage return. The **–Direction** line must then be held high for at least 7 microseconds before it may be brought low again, signaling the start of the data transmission.

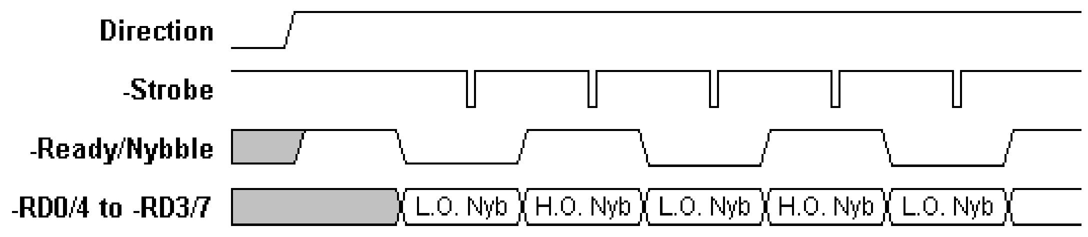
Once the **–Direction** line has been brought low to signal the start of the data transfer from the computer to the programmer, it must not be brought high again until after the last byte of data has been sent. When the **–Direction** line goes high, the programmer will assume that there is no more data forthcoming and will issue its prompt or error return at that time and will begin to wait for a new Remote mode command.

Receiving data or status from the programmer

During normal operation of the programmer, the **–Direction** line will, most of the time, be held high by the computer. While the **–Direction** line is high, the programmer may send status information to the computer. Normally, the programmer will only send status or information to the computer in response to a command sent by the computer.

Figure 4 below shows the rough appearance of a typical response sent from the programmer to the computer. In this case, it is a three-byte return.

Figure 4. Typical waveform of data transfer from the programmer to the computer



The indication that the programmer has information available for the computer is the state of the **–Ready/Nybble** line. Within a few nanoseconds of the **Direction** line going high, the **–Ready/Nybble** line will become active. At this time, it will usually be high, indicating that no data is ready from the programmer.

When status information or data is available from the programmer, the **–Ready/Nybble** line will go low. This will indicate that the low order nybble of the first data byte is available on the **–RD0/4** to **–RD3/7** lines. Note that the state of the **–Ready/Nybble** line changes at the same time as the **–RD0/4** to **–RD3/7** lines. There is, therefore, no setup time between when the data becomes available and the **–Ready/Nybble** line indicates that data is available.

Immediately after the **–Ready/Nybble** line goes low, the computer may pull the **–Strobe** line low, indicating to the programmer that the data on the **–RD0/4** to **–RD3/7** lines has been read by the computer. Only the high to low transition of the **–Strobe** line is important to the programmer, so the **–Strobe** line may be brought high again a few nanoseconds after it was brought low, depending on propagation delays, etc. Within a few microseconds of the **–Strobe** line high to low transition, the high order nybble of the first byte will be placed on the **–RD0/4** to **–RD3/7** lines and the **–Ready/Nybble** line will go high. At this point, the computer may read the high order nybble data from the **–RD0/4** to **–RD3/7** lines. When the computer has read this information, allowing the assembly of the complete first byte, the computer must bring the **–Strobe** line low, then high again to indicate this.

This completes the transmission of the first byte from the programmer to the computer. If another byte is available, the **–Ready/Nybble** line will once again go low.

The computer may bring the **–Strobe** line low any time after it has read the current nybble from the programmer. This high to low transition may occur any time from immediately after the transition of the **–Ready/Nybble** line, to days later. There is no timeout. The data will be held on the **–RD0/4** to **–RD3/7** lines until the high to low transition of the **–Strobe** line. After the **–Strobe** line transitions from high to low, there is usually a delay of a few microseconds before the next change of the **–RD0/4** to **–RD3/7** and **–Ready/Nybble** lines.

Note: When bringing the ***Direction*** *line low after receiving data from the programmer, the high to low transition of the* ***Direction*** *line must occur at least 7 microseconds after the last high to low transition of the* ***–Strob****e line. Failure to allow at least 7 microseconds between the last strobe of a nybble read and the start of a new command could cause the programmer's parallel port state machine to get out of sync with the computer. If this happens, a garbage character may be inserted in the upcoming command before sync is restored.*

Powering up the programmer

Data I/O programmers are able to determine if a serial connection has been made to either the Terminal or Remote ports even if no data is being transferred on the connection. Unfortunately, this is not true of the parallel port. The programmer is unable to tell if a computer is connected to the parallel port. It can, however, tell if commands are being sent to the programmer on the parallel port because in order to do that, the 5mA current source on the **Direction** line must be pulled to ground.

When the programmer powers up, it looks to its ports to determine which mode it will come up in. If it sees something connected to its terminal port, it will come up in terminal mode. If it only sees something on its Remote port, it will come up in Remote mode and expect commands from the serial Remote port. If nothing is connected to either of these serial ports, it will wait until either something becomes connected to one of the serial ports, or a command comes into the parallel port. If a command comes in through the parallel port, the programmer will go into Remote mode and expect commands from the parallel port.

This means that if you have written your own driver, have only the parallel port connected to your programmer, and are waiting for the Remote mode prompt (>) to come to you before sending any commands, you could be waiting a very long time.

If you have something connected to the Remote port on the programmer as well as the parallel port and are waiting for a prompt to come from the parallel port, you could again be waiting for a very long time. The programmer will issue its prompt to the serial Remote port and expect commands to come from there. If you want to control the programmer through the parallel port and you have both the Remote port and parallel ports connected to your computer, you must first issue a **77]** command to the programmer through the serial port. This will cause control to transfer to the parallel port. The parallel port on your computer will receive its prompt and you are ready to go.

If, however, you are powering on your computer with only the parallel port connected, there is a special procedure you should follow in order to get the Remote mode prompt that tells you the programmer has booted and finished its self-test.

The best way of accomplishing the power up of the programmer is to continually send escape characters to the programmer until it responds. This is probably best accomplished by pulling the **Direction** line low, presenting the escape character (0x1B) on the **+TD0** to **+TD7** lines, then toggling the **–Strobe** line low for at least a few nanoseconds. Do not wait for the **–Ready/Nybble** line to go low before toggling the **–Strobe** line. Remember, the programmer may still be running boot and test software and may not have time to look at the parallel port just yet.

Once the escape character has been strobed into the parallel port, keep the **Direction** line low until either the **–Ready/Nybble** line goes low, or a timeout of your choosing has passed. A good timeout value might be two milliseconds. After seeing the **–Ready/Nybble** line go low or waiting for the full timeout, bring the **Direction** line back high and watch the parallel port for the Remote mode prompt character. If the Remote mode prompt character (>) does not come back within some timeout period (again, two milliseconds would be good), then pull the **Direction** line low and strobe another escape character into the programmer, waiting again until either the **–Ready/Nybble** line goes low or the timeout period has expired. Repeat this operation until the programmer has either come up, or until so much time has elapsed that you suspect a problem with the parallel interface or the programmer.

Discontinuing a device operation

Canceling a device operation is accomplished with the parallel port much the same as it was done with the serial port. Just send an escape (0x1B) character to the programmer. This consists of bringing the **Direction** line low, presenting the escape character to the **+TD0** to **+TD7** lines, toggling the **–Strobe** line, then waiting for the **–Ready/Nybble** line to go low before bringing the **Direction** line back high. At this point, the programmer should respond with the Remote mode prompt (>) character.

Discontinuing a data transfer to the programmer

The computer has more control of a data transfer through the parallel port than it does through the serial port. With the serial port, the computer must stop sending characters and then wait until the programmer has figured out that you've stopped sending characters, which will mean the computer will be waiting for some timeout period.

With the parallel port, in order to stop a transfer to the programmer, simply pull the **Direction** line high. The programmer will know immediately that the transfer has stopped and will shortly issue the Remote mode prompt (>) character.

Discontinuing a data transfer from the programmer

Causing a halt to a data transfer from the programmer to the computer is accomplished in much the same way as canceling a device operation. Simply send an escape to the programmer. This is done by pulling the **Direction** line low, presenting the escape character (0x1B) to the **+TD0** to **+TD7** lines, toggling the **–Strobe** line low for at least a few nanoseconds, then waiting for the **–Ready/Nybble** line to go low before bringing the **Direction** line back high again.

This should terminate the data transfer and cause the Remote mode prompt (>) to be sent to the computer.

The only restriction on this operation is that the **Direction** line must not go low before 7 microseconds have elapsed since the last high to low transition of the **–Strobe** line. To pull the **Direction** line low too soon could result in throwing the parallel port communications temporarily out of sync.

Data transfer timeouts

Your computer interface program is king. There are no timeouts. You could make the programmer wait for days between characters, if you like. It will wait.