

Pin-Master 48



Universal Programmer



Pin-Master 48

UNIVERSAL DEVICE PROGRAMMER

USER MANUAL

MQP Electronics
Park Road Centre
Malmesbury
SN16 0BX
England

Tel: +44 (0)1666 825666
Fax: +44 (0)1666 825141

e-mail: sales@mqp.com
Website: www.mqp.com

Issue 1 - 11/1996
Issue 2 - 06/1999
Issue 3 - 12/2000
Issue 4 - 05/2001

Table of Contents

Introduction.....	1
Overview	1
Setting Up Your Programmer	1
Description of PM48 Programmer	1
Power Requirements	1
Voltage Setting.....	1
Earthing	2
Mobile Operation	2
Computer Requirements.....	2
Earth Bonding Point.....	2
JTAG/ISP Connector.....	2
Switching on.....	2
Software Installation.....	3
Operating Your Pin-Master 48	3
Selecting a Device Type.....	3
Manual Selection.....	3
Auto Selection.....	4
Programming a Device	4
To Program a Sequential Set of Devices.....	4
To Program in Split Mode.....	4
Programming Part of a Device.....	4
Append Programming	4
Manual Programming	4
Reading a Device	4
Checking a Device is Blank	4
Calculating Checksum.....	4
Verifying a Device against a File	5
Special Functions	5
Serial Numbering	5
Adjustments to Settings.....	5
Script (Batch) Mode	7
Introduction.....	7
Screen elements of Batch mode presentation.....	7
Promdriver Enhanced Batch Mode Specification.....	7
Syntax.....	7
Control Flow - Labels	7
Control Flow Commands	8
Conditionals	8
Error handling	8
The NO_STOP Instruction.....	8
Command Description.....	8
Example Script	17
Alphabetical Command List.....	19
String Variables	20
Truth Variables.....	20



Appendix A..... 21
 Device List 21
Appendix B..... 22
 Support Arrangements..... 22
 Software Support 22
 Registration of a new Programmer..... 22
 Activation of further year's support..... 22
 Web Page 22
 Information about support status..... 22
 Repair Service outside Warranty..... 22
 Hardware Service Contract 22
 Transporting the Programmer 22
Appendix C..... 23
 File Formats..... 23
Appendix D..... 27
 Warnings 27
Appendix E..... 28
 Frequently Asked Questions 28

Description of PM48 Programmer

Introduction

Overview

This manual describes in detail the operation of the Pin-Master 48 programmer.

Setting Up Your Programmer

IF YOU DON'T READ THE REST OF THE MANUAL AT LEAST READ THESE INSTRUCTIONS

1. Unpack programmer and leads.
2. Before connecting to the mains supply, check that the correct voltage is indicated on the label on the back of the unit.
3. Connect the D-type communications lead to the programmer socket and a PC parallel printer port.
4. Switch on the programmer - it will go through its start up sequence, flashing the LEDs and stopping with just the power (green) light on.
5. Insert the CD into the drive and install. Follow the installation instructions.
6. Run PROMDRIVER. The first time this program is run, you will be required to complete the initialisation details (telling PROMDRIVER which parallel port to use).
7. PROMDRIVER will now automatically register your new programmer.
8. You are now free to use the programmer.



The PM48 programmer is controlled by an internal microprocessor. On power up the software is downloaded from the PC to the programmer. Software upgrades only involve installing the latest version onto the PC. The design features re-configurable logic, and allows: any voltage on any pin, power supplies on any pin, programmable slew rates, and voltage measurement of any pin.

Power Requirements

Operating Voltages:	110/220/230/240V a.c.
Frequency:	47 - 66Hz
Power:	25VA

Voltage Setting

The programmer can be switched between 110V and 220/240V by the following procedure:

- Switch off the unit and disconnect the mains lead.
- Remove the four screws securing the lid.
- Remove the lid by raising the rear and lifting it forward. The lid can be rested on its right hand edge without disconnecting the LED cable.
- Exchange the position of the transformer cable plug and adjacent dummy plug on the 230/110V connectors.
- Replace the lid and four screws.
- **IMPORTANT**: MARK THE UNIT WITH A LABEL SHOWING THE NEW VOLTAGE IN SUCH A WAY THAT

THE ORIGINAL VOLTAGE LABEL IS COVERED.

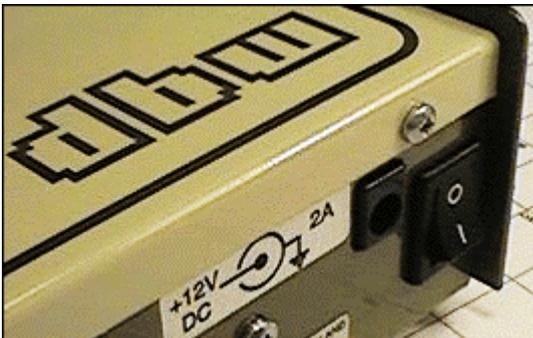
Earthing

THIS EQUIPMENT IS SAFETY CLASS 1 AND MUST BE EARTHED. We highly recommend that you use the mains lead supplied.

Mobile Operation

To allow mobile operation, the Pin-master 48 may be powered from a 12V d.c. power supply (e.g. a car battery). In this case a safety earth is not required. Use the 2.5mm standard length (10mm shaft) d.c. connector provided. You should be aware that the programmer metalwork is in this case connected to the negative line of the d.c. supply. This is normally the potential of a car chassis. **DO NOT USE IN VEHICLES WITH POSITIVE EARTH SYSTEMS.**

Operating Voltage: 11 - 15V d.c.
Power: 25 VA



To avoid possible damage to the programmer or the portable PC, always connect the parallel interface lead before connecting the power lead to the programmer.

Computer Requirements

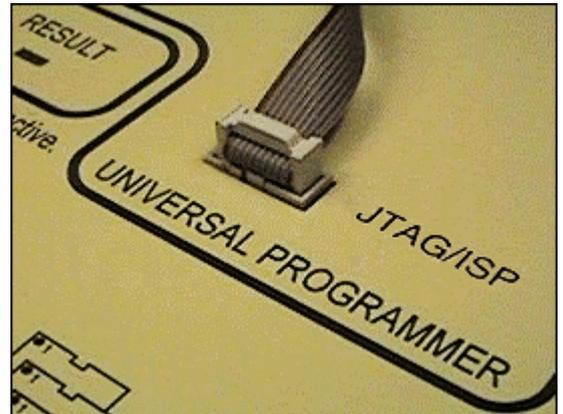
The supplied software will run on any PC compatible with a hard disk and a spare parallel port. No expanded or extended memory is required. PROMDRIVER software when installed will take up about 10Mbytes of your hard disk.

Earth Bonding Point



The earth bonding point can be used to plug in an anti-static wristband conforming to EN100 015-1.

JTAG/ISP Connector



Many of the larger programmable logic devices are now being designed with JTAG compatibility and in system programming. This connector has been provided to facilitate this on-board programming (see the application notes on the CD to see what devices are suitable for ISP programming).

Switching on

The programmer will operate its sequential self-test software. The LEDs will flash in a pre-determined pattern and stop with the POWER light (green) only on. If a fault is

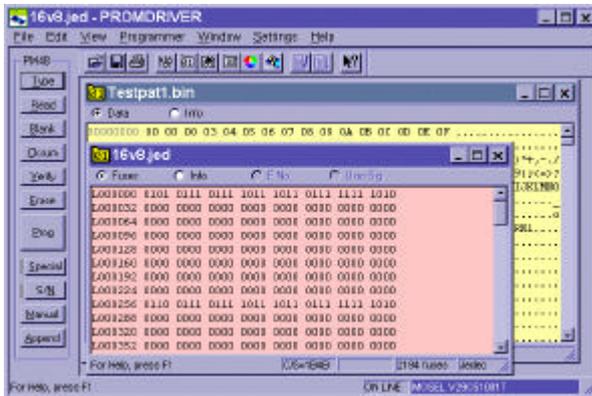
detected on start-up, the LEDs will stop with one or more of the other LEDs on. If this occurs please note which LEDs are on and notify MQP Electronics or your distributor.

Software Installation

Firstly you need to install the files from the CD provided on to your hard disc drive. **We do not use any form of copy protection, create any hidden files or modify your disc in any way.**

Please feel free to circulate this CD to your colleagues and friends as a demonstration of our software. There is a copy of our Website on this CD which contains datasheets on our package converters. If you have any comments about our products or software we would like you hear from you.

The first time of operation our Initialisation program will be run automatically, allowing you to select which parallel port your programmer is to be connected to, and to set-up a printer.



Operating Your Pin-Master 48

You will find the operation of PROMDRIVER software to be extremely intuitive.

During all operations involving a device in the socket, the 'ACTIVE' LED will be on continuously. PLEASE NOTE: THE DEVICE SHOULD NEVER BE REMOVED FROM THE SOCKET WHILE THE 'ACTIVE' LED IS ON.

The 'RESULT' LED will indicate success or failure of the any operation after completion:

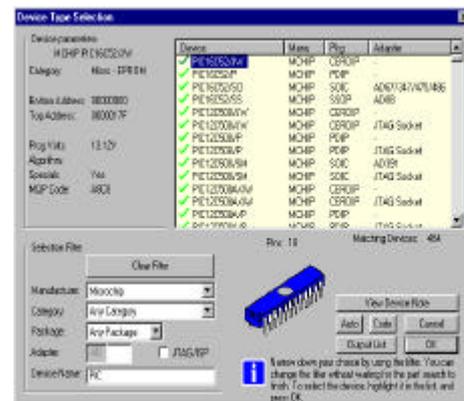
On	Success
Slow Flash	Operation Failed
Fast Flash	Mount Error
Short Flash	Socket Empty
Double Flash	ID Error

Selecting a Device Type

Manual Selection

Choosing the TYPE button opens MQP's quick and user friendly method of selecting the device that you want to read or program.

Device Type Selection Menu



Simply type in all or part of the device name and all matching devices are shown in a scrolling window. Highlight your exact choice and select it. To the left of the device screen, full details about the device type, size,

algorithm, etc. as well as the internal 'MQP Code' are displayed.

Specifying the manufacturer of the device or the device category will reduce the number of choices displayed.

The device may also be selected using the 'MQP Code'. These codes are also used in Script Mode, the MQP Production Language. A full or partial list of 'MQP Codes' can be printed to a file from the Device Type Selection menu.

Auto Selection

Your Pin-Master 48 can detect an internal Silicon Signature, which is embedded into the chip during manufacture and is usually permanent. Devices containing this feature include the more recent EPROMs (from Quick-pulse 2764 onwards), Flash devices and some microprocessors.

Programming a Device

Programming a single device from a file simply involves opening the file and pressing the Program button. The device in the socket will be blank checked if appropriate, programmed with the contents of the file and verified (at one of two voltages - depending on the adjustments set in the programmer). At each stage, a bar will be displayed on the screen indicating progress.

To Program a Sequential Set of Devices

If the file is larger than the size of the selected device, the software will assume that you want to program a set of identical devices. After the first device has been programmed you will be requested to insert the second device in the set and so on until the whole set of devices have been programmed.

To Program in Split Mode

The sequential bytes of an object file are normally programmed sequentially into the EPROM. If an EPROM set for a microprocessor with a 16-bit data bus is to be made, it may be necessary to program alternate bytes of a file into an EPROM. The

BYTE SELECT function in the Current File Setting menu achieves this by allowing ALL, ODD or EVEN bytes of a file to be programmed into the EPROM.

Note that ODD and EVEN refer to the addresses within the file.

The 'ALL bytes (REVERSED)' selection is used for 16 bit wide EPROMs when the order of the bytes within a word is not the standard order for the file format selected.

The Byte Select setting will affect Programming and Verification only.

Programming Part of a Device

Append Programming

This button allows:

- a single device which may already be partially programmed to be further programmed,
- a device to be programmed between specific addresses only, from any address within a file.

To achieve this no blank check is performed and verification is limited to individual bytes as they are programmed. Please note that attempts to program the erased value into an already programmed location will be ignored (no error message will result).

Manual Programming

This button allows individual addresses in the device to be programmed one byte or word at a time.

Reading a Device

This is achieved by using the Read button.

Checking a Device is Blank

Choose the Blank button and you will be told whether or not the device in the socket contains its blank (unprogrammed or erased) value at all locations.

Calculating Checksum

The checksum of a device or file is calculated by adding up all the bytes or words of the device or file. This is a useful method of identifying similar versions of devices, as it is

unlikely that different programs will produce the same checksum.
 Select CHECKSUM and the checksum of the device currently in the socket will be displayed.

Verifying a Device against a File

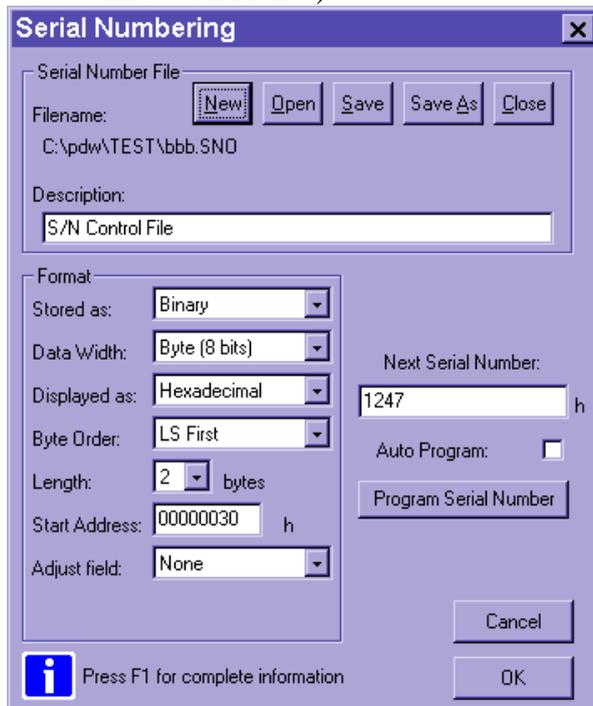
This will show any differences between the contents of the device in the socket and the open file.

Special Functions

Some EPROM, EEPROM, Flash and Microprocessor types can be security programmed to prevent unauthorised access to data in the device or overwriting the contents of part or all of the device. The correct device type must be selected before the Special button is pressed.

Serial Numbering

Each device may be programmed with an automatically incrementing serial number, to your specification. You should ensure that the area in the device which is to contain the serial number is left unprogrammed (for most devices filled with FF hex).



In order to maintain a constant checksum of all the serial numbered devices, the

'INVERSE' of the serial number can be programmed in a second area of the device. This will also need to be left blank. Serial numbering can take place as a separate operation (MANUAL), or as a part of the normal remote PROGRAM operation (AUTO).

Before starting to input your serial number requirements, you should ensure that the correct device type is selected. This allows address range checking to take place, and does not limit your final choice of actual device type.

Adjustments to Settings

By selecting PROGRAMMER SETTINGS, various settings can be modified to suit your requirements and those of the particular device you are programming.

Blank Check

This function checks that the device is blank before programming. Disable if not required, e.g. for EPROM emulators.

ID Check

If enabled this will check the silicon signature of devices before they are operated on. If there is a mismatch a warning may be given. Typically on an EPROM the ID check involves applying 12V to A9 on the device. If you are programming a Microprocessor which programs like an EPROM using an adapter, be sure to turn off ID checking to avoid damaging the microprocessor.

Mount Check

This does not stop a check for the presence of a device, but allows programming to be attempted even if no device is detected in the socket.

Reverse Check

Checks for the orientation of the device before VCC and VPP are applied. Some device types may trigger this precaution even when correctly inserted, so this check can be disabled.

Verify Level(s)

Programmers sometimes verify just at the nominal VCC level. However to ensure that your devices have been completely programmed, the Pin-Master 48 allows you to

verify at any two VCC levels between 2.75V and 6.75V or, by selecting any of the following: VCC +/- 0%, 5%, 10% and 20%.

Buffer Clear Value

This setting determines the value programmed into undefined parts of a device. For example, if you want to program a 64 Kbits device with a file that is 32 Kbits long, normally the second half of the device will be left containing FF hex (the erase value of the device). However it is possible to change the Buffer Clear Value to 00 hex if necessary. Note that this will affect the checksum of the programmed device.

Access Time

When programming certain devices, (usually older types or microprocessors which program like an EPROM using an adapter) it is necessary to slow down the access time in order to ensure correct verification.

Script (Batch) Mode

Introduction

PROMDRIVER has been enhanced to provide a comprehensive script mode to streamline the use of the programmer in a production department.

This allows the user to write a script to automate programming jobs. For example the file can be downloaded, and the correct device type selected automatically. The operator can then be prompted to insert devices as required. A running total of Programmed devices, Failures, and Quantity still required may be displayed. Other features include user-defined messages, user-defined menus, and setting programmer functions. This chapter is a definition of the Batch Production Language.

Screen elements of Batch mode presentation

Heading

By default the programmer manufacturer's logo will appear at the top of the screen, occupying 6 lines (4 available for text). By using the HEADING command the user can customise or remove this heading.

Operator Instruction Area

This is a mandatory area of the screen capable of 3 lines of text display (this may be extended if necessary). It will be coloured in a manner to draw attention.

Menus or messages

The area occupied by the operator instruction window is also used for messages or menus, however the message or menu will actually be another window opened over the top of the instruction area, with a minimum size of 3 lines of text.

'Failures', 'Programmed' and 'Required' boxes.

These boxes each have their own standard location at the bottom of the screen in an appropriate colour.

Failure – RED

Programmed – GREEN

Required - BLUE

Promdriver Enhanced Batch Mode Specification

Like most programming languages the PROMDRIVER Production Language uses English words only, even on other language versions of the software. The user interface will still be in the required language, but the script language itself always uses English keywords.

All scripts will be held in non-document (plain ASCII) files, e.g. SCRIPT.SCR. There is no special restriction of the filename.

Syntax

Commands normally appear in the file one per line. Some special commands spread over several lines. Comments are permitted; introduced by a semicolon, and finishing at the end of the line.

Any keyword may include the underscore character (`_`) at any point in the word. This is ignored by the parser but may improve readability. e.g. BYTESELECT and BYTE_SELECT are equally acceptable.

Control Flow – Labels

A label is always preceded by a colon in the first column, and is the only item on the line (apart from any comments). References to labels are not preceded by a colon. Labels are any length with 16 significant characters. Upper and lower case are equivalent. Labels may contain alphanumeric and the underscore `'_'`.

Example:

```
:START_AGAIN_12  
GOTO START_AGAIN_12  
(actually an example of an infinite loop)
```

Control Flow Commands

GOTO <label>

This instruction is self-explanatory but there are some restrictions placed on its use to avoid the usual pit-falls of programming with GOTO's.

Conditionals

There are 2 structures for 'if' type conditions, one for statements including GOTO's, and one for other statements.

IF <condition> GOTO <label>

The GOTO on a condition must be on the same line as the IF as shown. No ENDIF or ELSE are required or allowed.

```
IF <condition>
<commands>
[ELSE
<commands>]
ENDIF
```

GOTO is not allowed within a conditional section.

Typical conditions are:

IF YES

IF NO

IF ESC

IF OTHER_REPLY

IF MORE_NEEDED

IF ERRORLEVEL = <n>

where <n> is a decimal integer.

Error handling

Each function having the possibility of an error (e.g. Programming, Append, Read etc) will set a (16 bit) variable called ERRORLEVEL. 0 will signify success, and other values will have fixed predetermined meanings (e.g. DEVICE NOT PRESENT,

DEVICE NOT BLANK, USER ABORTED ETC). ERRORLEVEL can be tested after each operation using an IF construction (see above).

If a syntactical error is detected in a batch command, it is reported on the screen (with the erroneous line number) and processing is immediately aborted.

The NO_STOP Instruction

In order to speed up operations etc., NO_STOP can be added to certain commands and as long as no error occurs during the command, PROMDRIVER will not stop and display a message at the end of an operation. One use of this would be if a PROM is being programmed from a number of files using append mode. Operator intervention may not be required after each Append Command if programmed correctly. If NO_STOP is permitted within a command, the position it must be placed will be indicated in the following list.

Command Description

TYPE <device_code>

TYPE AUTO [NO_STOP]

TYPE ?

TYPE AUTO is the same as AUTOKEY (see below).

ERRORLEVEL codes set

0 = type selected ok

1 = user aborted selection

2 = type not recognised

TYPENAME "<text_string>"

Once a device type has been selected, see above, its name can be changed to whatever the user wants.

This can be useful when programming microprocessors that program like EPROMs. First the TYPE is set using the EPROM's Device code, then using TYPENAME, the text string in the status area can be corrected to display the microprocessor's actual name.

For other special commands for microprocessors that program like EPROMs see section on 'Frequently Asked Questions'.

FORMAT <file_format>

where <file_format> is

BINARY

INTHEX

MOTHEX

INTOBJ80

INTOBJ51

INTOBJ86

INTOBJ286

FREEHEX

FREEDEC

TEXTAG

TEKHEX

TEKHEXEXT

MITHEX

ERRORLEVEL codes set

0 = set ok

FILEADDRESS <fileaddress>

where address is a hex number between 0 and FFFFFFFh

ERRORLEVEL codes set

0 = set ok

POLARITY <polarity>

where <polarity> is TRUE or INVERTED

ERRORLEVEL codes set

0 = set ok

BYTESELECT <byteselect>

where <byte_select> is:

NORMAL

REVERSED

ODD

EVEN

ERRORLEVEL codes set

0 = set ok

NOEXIT

Currently some actions cause an immediate exit from batch programming. This would include the user pressing ESC. This may not be desirable, so this command is available to prevent accidental exit from batch mode. Exit can then only be by programmed means. In order to allow a means of exit in the case of an infinitely looping program, ALT/F10 will allow exit.

EXIT

Terminates batch file execution.

WAIT n

Delays for n seconds.

SPECIAL [NO_STOP] MANUAL <command> [<hex_value> [<hex_value>]]

SPECIAL [NO_STOP] AUTO <command> ON [<hex_value> [<hex_value>]]

SPECIAL [NO_STOP] AUTO <command> OFF

Special functions such as 'Lock PROM' can be found for some devices in the DEVICE SPECIALS selection in menu mode. Most of these functions are also available in batch mode.

There are two ways of activating each special function similar to menu mode and stand-alone mode. MANUAL activates the special function at the time it is encountered in the script file, whereas when an AUTO command is given the special function is activated every time programming occurs.

The appropriate <command> is the item number to the left of the menu or following 'S' in the square brackets to the right of the menu. It will depend on the device concerned. For example, with an 8751 the only function available is 1 which blows the security bit.

To find out which functions are available look at the menu mode SPECIALS with the appropriate device selected. The help screen (F1) will give further information.

Available parameters are indicated in square brackets []. These are either a hex value, e.g. in the case of a PIC micro-controller's ID, or the item number in a subsequent menu, e.g. for PIC oscillator setting of XT, the <hex_value> should be set to 3.

ERRORLEVEL codes set

0 = Set ok

1 = Function programming failure

AGAIN?

AGAIN? GOTO <label>

Displays message asking if a repeat is required. If reply is YES or ENTER, starts executing batch mode instructions from the start, or from the label.

MENU

“<menuline_1>” [GOTO <label> | TYPE <device_code> | NOOPTION]

“<menuline_2>” [GOTO <label> | TYPE <device_code> | NOOPTION]

...

“<menuline_n>” [GOTO <label> | TYPE <device_code> | NOOPTION]

MENUEND

Displays a menu. On selection of a line with an associated GOTO, goes to that line. On selection of any other line goes to next instruction in file after MENUEND.

If line text is followed by TYPE <device_code> then this is processed as a normal TYPE setting command.

If a line is followed by NOOPTION it is only displayed but not offered as an option, so the cursor cannot be moved to such lines. This is a useful command for headers and other explanatory lines.

GOTO and TYPE may both appear on the same line but NOOPTION may not be accompanied by either.

DELETE <filename>

Deletes the file specified without asking the user. No action if file does not exist.

**RENAME <old_filename>
<new_filename>**

Renames the file specified without asking the user. No action if file does not exist.

%REQUIRED

%PROGRAMMED

%FAILURES

%TYPENAME

%FORMAT

%FILEADDRESS

%POLARITY

%BYTESELECT

%SERNO

String variables which substitute the value of the variable into a text string, i.e. inside a pair of quotes (“), of a heading, message or menu.

BLANKCHECK ON

BLANKCHECK OFF

Defaults to ON

IDCHECK ON

IDCHECK OFF

Defaults to ON

MOUNTCHECK ON

MOUNTCHECK OFF

Defaults to ON

REVCHECK ON

REVCHECK OFF

Defaults to ON

STATUS ON**STATUS OFF**

Enables, or disables, the display of current settings like file name, file type, bytes select, polarity, file address, device type, adapter required, etc. Default is OFF.

HEADING

"<heading_line_1>"

["<heading_line_2>"

["<heading_line_3>"

["<heading_line_4>"]]]

HEADINGEND

If HEADING is not specified the batch window will show the default heading information block. Using HEADING replaces this with user defined information, which may be null information, if there are no lines between HEADING and

HEADINGEND. Any message lines after line 4 will be ignored. HEADING must be the first actual command in the response file, if the default heading is to be completely disabled. The heading may be changed, at any time in the script.

MESSAGE [GET_YES_NO]

"<message_line_1>"

["<message_line_2>"

["<message_line_3>"

["<message_line_4>"]]]

MESSAGE_END

Messages will appear in the message area. This area will be enlarged (by opening a window over the top) if the message height requires it. The message is displayed until any key is pressed.

If GET_YES_NO is included on the MESSAGE line, the relevant parameters (YES, NO, ESC or OTHER_REPLY) are set when the key is pressed and these parameters can be later tested by the IF construction.

RUN <program_name> ["<argument>" [..."<argument>"]]

RUN_STDIO <program_name> ["<argument>" [..."<argument>"]]

Runs a user program with optional parameters. RUN will assume that the user program either does no screen I/O, or that it deals with restoring the screen itself. RUN_STDIO will open up a window (full screen) for user I/O, and then clear it away afterwards.

The user program may return a code which will be placed into ERRORLEVEL.

FAILURES 0

FAILURES OFF

FAILURES ON

This causes a failures box to be displayed (or removed) or resets the number of failures to the number shown if the box is already displayed. The number of failures is automatically incremented by any 'Failure to program' messages in Program, Append, Serial Number Append, or Special Program operations, but not those caused by 'device not present'.

PROGRAMMED 0

PROGRAMMED +

PROGRAMMED OFF

PROGRAMMED ON

This causes a programming success quantity box to be displayed (or removed) or resets the number of successes to the number shown if the box is already displayed.

The number of successes is incremented by a PROGRAMMED + command, e.g.

```
PROGKEY <filename>
IF ERRORLEVEL = 0
    PROGRAMMED +
    REQUIRED -
ENDIF
```

REQUIRED 20

REQUIRED -

REQUIRED ?

REQUIRED OFF

REQUIRED ON

This causes a devices required quantity box to be displayed (or removed) or resets the number required to the number shown if the box is already displayed.

The number required is decremented by a REQUIRED - command.

VERIFYLEVEL <setting>

where <setting> is:

+/-0%

+/-5%

+/-10%

+/-20%

Verification of programming may be performed at VCC +/- 0%, 5%, 10% and 20%.

VERIFY_LOW <setting>

VERIFY_HIGH <setting>

where <setting> is a valid VCC value.

EMPTYFILL <setting>

where <setting> is:

0

FF

DEFAULT

This defines the hexadecimal value that will be used to fill parts of the device that are not defined in the file to be downloaded, displayed or printed. It will normally be set to the blank value for the currently selected device (DEFAULT), but you may want to force it to another value so that, for instance, the checksum is consistent with another manufacturer's programmer.

ACCESSTIME <setting>

where <setting> is:

NORMAL

SLOW

By setting the access time to SLOW, this function allows you to increase the access time for reading, verifying and checksumming the selected device. This is necessary for certain devices especially microprocessors that program like EPROMs but in different packages. If set to SLOW, the setting will be displayed in the status area.

The default setting is NORMAL.

SERNO FILE <file_name>

SERNO AUTO

SERNO OFF

SERNO CHANGE

These functions allow the serial numbering of programmable devices to be carried out automatically each time that they are programmed. SERNO FILE <file_name> must be processed in the script file before any of the other SERNO instructions can appear. <file_name> refers to the serial number filename (suffix .SNO) that has been set up in menu mode.

SERNO AUTO and SERNO OFF are equivalent to auto and manual in menu mode. SERNO CHANGE allows changes to the next serial number to be programmed from within batch mode.

NEW

Puts all settings as they were when batch mode first started.

PROGKEY [NO_STOP] <filename>

PROGKEY [NO_STOP] ?

PROGKEY [NO_STOP] ??

This instruction causes a device (or set of devices) to be programmed. The user will be prompted to insert or remove devices and informed of progress.

APPEND [NO_STOP] <prom start addr.> <prom end addr.> <filename> <file start addr.>

Allows the menu mode append function to be automated. All parameters must be specified (no ? user prompts). Example:

APPEND 1000 1fff prog.hex 3000

will cause the PROM addresses 1000 to 1fff hex. to be programmed with data from prog.hex starting at address 3000 hex. in the file (and ending at 3fff).

Note: For PICs, worldwide EPROMs and other devices with widths greater than 8 bits, all addresses referred to, are word addresses (not byte addresses).

READKEY [NO_STOP] <filename>

READKEY [NO_STOP] ?

READKEY [NO_STOP] ??

Allows the menu mode read function to be automated.

ERASEKEY [NO_STOP]

Same as pressing the ERASE key.

VERIFYKEY [NO_STOP] <filename>

Same as pressing the VERIFY key.

CKSUMKEY

Same as pressing the CKSUM key.

BLANKKEY [NO_STOP]

Same as pressing the BLANK key.

AUTOKEY [NO_STOP]

Auto type select.

Example Script

This is a printout of an example of a user generated script file called:

SAMPLE1.BAT

The actual script follows, and contains several comments to explain the use of commands. The total effect of the script is to produce a number of EPROMs for a particular job, allowing for a choice of EPROM type, either from a predetermined list or by auto-selection.

The correct file will be automatically downloaded, and a count is maintained on screen of programmed devices and failures. The user is also asked to say how many he is required to blow, and the number still required is displayed.

We suggest that you use a copy of this file to modify for your own purposes until you become familiar with the programming language.

Sample Script 'SAMPLE1.SCR'

; sample batch program to blow some 27C010s
; programming commands are shown in UPPER CASE for clarity although
; they may equally well be in lower case.

; if first command is HEADING then the default logo
; will not appear

HEADING

“ ACME MANUFACTURING CORP.”
“”

“ Program to blow some 27C010s for the Mega-Cyclatron Job”

HEADINGEND

; allow operator to pick one of the available parts
; lines with no_option cannot be selected

MENU

“ As Acme Co. stores may provide different EPROM types please select the” NO_OPTION

“ type available from the following menu” NO_OPTION

“” NO_OPTION ; example of a line which can't be chosen

“ AMD 27C010” TYPE 99A9 ; example of a menu type setting

“ Intel 27C010” TYPE EDE8

“ ST 27C010” TYPE 8889

“ NS 27C010” TYPE C013

“ OTHER” GOTO AUTOSELECT ; example of a menu goto

“ (use Cursor Up and Down, then press ENTER)” NO_OPTION

MENU_END

; wait command causes delay for a number of seconds

WAIT 1

GOTO DOJOB ; if we're here we must have selected a type

:AUTOSELECT ; label to goto if type wasn't in list
; tell operator to put in a part to identify

MESSAGE

“Place one of the devices in any socket of the programmer”

“in order to automatically determine its type”

“ ...when ready press any key”

MESSAGE_END

; now do an auto select operation

TYPE AUTO

; demonstrate that we know the type now

MESSAGE

“The type of device you wish to program is %type_name”

```
“      ... press any key”
MESSAGE_END
      ; in all cases we get here to blow the devices
:DOJOB
      ; first we download the file required
FORMAT BINARY
      ; turn on the box showing how many successful
      ; programmed devices we get
PROGRAMMED 0
      ; and a box showing failures
FAILURES 0
      ; ask the operator how many he needs to do
      ; and open up a box with this number
REQUIRED ?
      ; this label is where we'll loop back to,
      ; to program more parts
:LABEL
      ; tell operator to put a blank part in
MESSAGE
“”
“      Put blank %type_name part in socket and press any key”
“”
“      = (Press ESC to exit when done) = “
MESSAGE_END
      ; do some programming
PROG_KEY SAMPLE1.SCR
      ; add success to programmed message box
PROGRAMMED +
      ; decrement number if failed
REQUIRED -
      ; ... and loop
GOTO LABEL
      ; in this file operator decides when he has finished and exits by
      ; pressing ESC.
```

Alphabetical Command List

ACCESSTIME	Slow down the read access time
AGAIN?	Repeat script from start if user answers 'yes'
AGAIN? GOTO	Repeat script from label
APPEND	Activate Append function
AUTOKEY	Do remote auto type set
BLANKCHECK	Turn automatic blank check on or off
BLANKKEY	Do a remote blank check
BYTESELECT	Set which bytes are selected during download
CKSUMKEY	Do a remote checksum operation
DELETE	Delete an MS-DOS file
EMPTYFILL	Over-rides blank value of device if required
ERASEKEY	Do a remote erase operation
EXIT	Exit script
FAILURES	Turn on/off 'Failures' box
FILEADDRESS	Set default start address in files
FORMAT	Set file format
GOTO	Start executing from a label
HEADING, HEADINGEND	Define a screen heading
IDCHECK	Turn automatic id check on or off
IF , ELSE, ENDIF	Condition construction
MENU, MENUEND	Display a menu and get response
MESSAGE, MESSAGE END	Display a message (optionally get response)
MONITORCKSUM	Validates the checksum before programming
MOUNTCHECK	Turn automatic mount check on or off
NEW	Resets all parameters
NOEXIT	Prevent ESC key from exiting script
POLARITY	Set data polarity for file downloads
PROGKEY	Program the device
PROGRAMMED	Turn on/off 'Programmed' box
READKEY	Read the device
RENAME	Renames the specified file
REQUIRED	Turn on/off 'Required' box
REVCHECK	Turn automatic reverse check on/off
RUN	Run a user program under MS-DOS
RUN_STDIO	Run a user program requiring use of screen and keyboard under MS-DOS
SER_NO	Serial numbering
SPECIAL	Manually or automatically program specials
STATUS	Turn on/off on-screen status information
TYPE	Set device type
TYPENAME	Change device type name
VERIFYKEY	Do a remote verify
VERIFYLEVEL	}
VERIFY_LOW	} Change the verify voltage range
VERIFY_HIGH	}
WAIT	Do nothing for defined number of seconds

String Variables

%BYTESELECT	expands to ODD, EVEN, ALL etc
%FAILURES	expands to number of failures
%FILEADDRESS	expands to default file address
%FORMAT	expands to description of file format
%POLARITY	expands to TRUE or INVERTED
%PROGRAMMED	expands to number of successes
%REQUIRED	expands to number required
%SER_NO	expands to next serial number
%TYPENAME	expands to description of device type set

Truth Variables

ERRORLEVEL = <n>	
ESC	true if ESC key was pressed
MORE_NEEDED	true if 'required' variable > 0
NO	true if reply was 'no'
OTHER_REPLY	true if reply was not 'yes', 'no', or ESC
YES	true if reply was 'yes'

Appendix A

Device List

The list of devices can be printed from the PROMDRIVER TYPE select menu.

Appendix B

Support Arrangements

- Software Support
- Repair Service Outside Warranty

Software Support

Software updates are available at any time from our web page. However devices added after the end of the your registered support period will not be selectable until you obtain a new registration code from us. This code will be provided free of charge on request (see below). Any user requiring extra upgrades and not having access to a modem will be asked to pay a handling charge.

Registration of a new Programmer

The programmer will automatically be registered for one year's software support, the first time it is switched on.

Specifically, follow the PROMDRIVER installation instructions. However you are not forced to use this particular computer to run the programmer. The programmer itself stores the registration information.

Activation of further year's support

To obtain a year's extra support you must supply the serial number of the programmer. You will receive a new registration number in return. Enter this number into the PROMDRIVER registration screen.

Web Page

See front page of this manual for the address.

Information about support status

The Info screen will give the unit serial number and the support date.

Repair Service Outside Warranty

Repairs are carried out on a 'return to base' basis, normally within 2-3 days. The price will be available on request.

If a programmer, returned for repair, is found to have an unreliable socket, caused by contamination, we will normally replace the socket, making a charge for this.

Hardware Service Contract

An annual service contract is available. Not covered are:

- faults caused by misuse
- replacement of sockets
- package converters
- carriage charges in either direction
- software upgrade support (see Software Support)

If a unit has a contaminated or damaged socket, the normal charge for refurbishment will apply before the unit can be repaired under the service contract.

Transporting the Programmer

Please observe the following advice if you need to transport your programmer:

- Always obtain a returns number first if returning a unit to us.
- If possible use the original purpose-made packaging.
- Pack any leads or other items returned separately.
- Remember that the programmer components are heavy and will undoubtedly receive rough handling in transit. Apply sufficient padding accordingly.
- Always send the programmer prepaid, or it will not be accepted.

Appendix C

File Formats

Object files up to 16 Mbytes in size are accommodated by PROMDRIVER. Files are permitted to be in any of the following formats. (Note that some versions of PROMDRIVER may omit certain filehex formats.)

Binary Image

This format is so called because each 0 or 1 required in the PROM is represented by a 0 or 1 in the file. The individual bytes of the file represent the individual bytes in the PROM in a one to one relationship. No address information is included in such a file. The file may be shorter than a PROM in which case PROMDRIVER will fill the rest of the PROM with erased byte value. It may be the same size as the PROM; or it may be larger than the PROM. In the latter case, bytes in excess of those required to fill a PROM are assumed to be intended for the next PROM in a set.

When programming or verifying, the first byte in the file is assumed to correspond to address 000000. This does not prevent the PROM being mapped at some different address in subsequent use, but merely provides a convenient way to refer to the contents of the file. The File Address setting function may be used to program or verify a PROM from any convenient point within the file. For example, setting the File Address to 002000, and the PROM type to 2764 would allow programming to occur starting with the (2000H)th byte in the Binary Image file, and thus relate to the second PROM in the set. When reading a PROM to a file the File Address offset has no effect on a Binary Image file. The number of bytes in the file will always equal the number of bytes in the PROM.

For normal use with PROMDRIVER the Binary Image format is recommended; as it involves less processing the system response is much faster.

Freeform Hexadecimal Format

This is a printable file format in which each byte is represented by two ASCII hexadecimal characters, which are separated from adjacent bytes by spaces or new lines. Like binary files no address information is included in the file.

PROMDRIVER generates such files with a space character after each byte and a <cr> <lf> pair after each 16 bytes, e.g.:

```
03 36 5F 3D C4 DD 02 B3 65 7D 91 04 D3 7A 56 8A
7C 45 F3 6D 74 28 D4 6E CD C3 C9 56 D6 94 3D 56
```

When reading in this format PROMDRIVER will regard sequences of valid hexadecimal characters as bytes (ignoring overflow) and any other characters (including new lines) as separators, thus:

1,2,3,c, d

01,02
03,0C, 0D

101:102:
103:F0C: 30D

are all equivalent.

Freeform Decimal Format

This format is the same as Freeform Hexadecimal except that up to 3 decimal characters are used to represent each byte.

Intel Hexadecimal

(Intel 8080 Hex ASCII and Intel 8086 Hex ASCII)

This format combines the Intel 8080 Hex ASCII and the Intel 8086 Hex ASCII formats. The latter is a superset of the former. PROMDRIVER reads both formats and generates the 8080 subset wherever possible. The Intel Hexadecimal format is a way of representing an object file in a format which may be directly displayed and read, as all characters within the file are printable ASCII characters, or <cr> or <lf> characters. Binary byte values are represented by a hexadecimal representation of the byte coded in ASCII. For example the eight bit value 01011100 is 5C in hexadecimal. To code this

in ASCII the 2 bytes 00110101 (ASCII for 5) and 01000011 (ASCII for C) are required. A hexadecimal file is made up of records. There are 4 possible record types:

Extended Address Record.

RECD MARK	REC LEN	ZEROS	REC TYP	USBA	CHK SUM
'.'	'02'	'0000'	'02'	'HHHH'	'CC' <cr> <lf>

Data Record.

RECD MARK	REC LEN	LOAD ADDR	REC TYP	DATA	CHK SUM
'.'	'HH'	'HHHH'	'00'	'HH...HH'	'CC' <cr> <lf>

Start Address Record.

RECD MARK	REC LEN	ZEROS	REC TYP	CS	IP	CHK SUM
'.'	'04'	'0000'	'03'	'HHHH'	'HHHH'	'CC' <cr> <lf>

End of File Record.

RECD MARK	REC LEN	ZEROS	REC TYP	CHK SUM
'.'	'00'	'0000'	'01'	'FF' <cr> <lf>

where H represents a hexadecimal character (0 1 2 3 4 5 6 7 8 9 A B C D E F) coded in ASCII, CC is a single byte (2 character) checksum which is the two's complement of the sum (modulo 256) of all the bytes (not characters) after the record mark but before the checksum in a record, and USBA is the Upper Segment Base Address; which is multiplied by 16 and added to the following record Load Addresses to produce the actual load addresses. USBA is assumed to be 0000 until an Extended Address Record is encountered. PROMDRIVER will read files containing any of these record types. When creating object files, it will never generate a Start Address Record, and will only generate an Extended Address Record if it is necessary. Therefore if a PROM address plus File Address offset exceed 00FFFFH then an Extended Address Record will be generated.

Hexadecimal files contain address information. When PROMDRIVER creates a Hexadecimal file it commences generating addresses from the value of the File Address.

Similarly when programming or verifying a PROM from a file, the Prom Offset must be set to the address within the file which corresponds to the first byte to be programmed into the PROM.

Intel hexadecimal files are limited to 1 Mbyte in size.

To allow compatibility with Digital Research Hex files a Record with a record length of 00 is regarded as an End of File Record.

Motorola Hexadecimal

(Exorciser and Exormax formats)

This format combines the Exorciser format and the Exormax format. The latter is a superset of the former. PROMDRIVER reads both formats and generates the Exorciser subset wherever possible.

Sign-on record.

START CHARS	BYTE COUNT	LOAD ADDR	DATA	CHK SUM
'S0'	'HH'	'HHHH'	'HH...HH'	'CC' <cr> <lf>

Data record (2 byte address).

START CHARS	BYTE COUNT	LOAD ADDR	DATA	CHK SUM
'S1'	'HH'	'HHHH'	'HH...HH'	'CC' <cr> <lf>

Data record (3 byte address).

START CHARS	BYTE COUNT	LOAD ADDR	DATA	CHK SUM
'S2'	'HH'	'HHHHHH'	'HH...HH'	'CC' <cr> <lf>

End of File Record.

START CHARS	BYTE COUNT	LOAD ADDR	CHK SUM
'S8'	'03'	'0000'	'FC' <cr> <lf>
'S9'	'03'	'0000'	'FC' <cr> <lf>

where H represents a hexadecimal character (0 1 2 3 4 5 6 7 8 9 A B C D E F) coded in ASCII,

CC is a single byte (2 character) checksum which is the one's complement of the sum (modulo 256) of all the bytes (not characters) after the start characters but before the checksum in a record, the BYTE COUNT is the total number of bytes following in the record, including the checksum.

PROMDRIVER will read files containing any of these record types.

When creating object files, it will never generate an S0 or S8 record, and will only generate an S2 record if necessary. Therefore if a PROM address plus File Address offset exceed 00FFFFH then an S2 record will be generated.

Hexadecimal files contain address information. When PROMDRIVER creates a Hexadecimal file it commences generating addresses from the value of the File Address.

Similarly when programming or verifying a PROM from a file, the Prom Offset must be set to the address within the file which corresponds to the first byte to be programmed into the PROM. Motorola hexadecimal files are limited to 16 Mbyte in size.

Intel 8080 Absolute Object

Both 8080 and 8086 Absolute Object formats are formed from a series of contiguous records each having the same basic format.

RECORD TYPE	RECORD LENGTH	RECORD CONTENTS	CHECK SUM
1 byte	2 bytes	(record length-1) bytes	byte

All bytes are represented directly in the file by their binary values. All multiple byte values are ordered least significant byte first. The CHECKSUM is the 2's complement,(modulo 256), of all the other bytes in the record.

In 8080 Absolute Object format PROMDRIVER is only concerned with the following record types. (These descriptions are not rigorous and only describe the records to the extent that they are used by PROMDRIVER.)

MODULE HEADER RECORD.

RECORD TYPE	RECORD LENGTH	MODULE NAME	MODULE NAME	RSVD	CHECK SUM
02	LLLL	LL	N...N	00 00	CC

CONTENT RECORD.

RECORD TYPE	RECORD LENGTH	SEG ID	OFFSET BYTES	DATA	CHECK SUM
06	LLLL	00	AAAA	DD..DD	CC

MODULE END RECORD.

RECORD TYPE	RECORD LENGTH	MOD TYPE	SEG ID	OFFSET	CHECK SUM
04	05 00	00	00	0000	F7

END OF FILE RECORD.

RECORD TYPE	RECORD LENGTH	CHECK SUM
0E	01 00	F1

where:

LLLL is 2 byte length.

AAAA is 2 byte load address for the data bytes.

DD is a data byte. The number of data bytes is only limited by the record length.

CC is the checksum byte.

For an absolute object record the Segment ID must be 0.

LL is a one byte name length.

N...N is a variable length ASCII string (the module name).

RSVD is a reserved field of 2 zero bytes.

PROMDRIVER responds to CONTENT records, and requires that the file ends with an END OF FILE record.

All other record types are ignored, except for illegal types which cause a file format error to be reported.

When creating object files PROMDRIVER generates:

- a Module Header record
- a variable number of Content records
- a Module End record
- an End of File record.

The MODULE NAME generated is MQP@ELECTRONICS

Intel 8080 Absolute Object files are limited to 64 Kbyte in size.

Intel 8086 Absolute Object

This format is made up of records having the same basic form as for 8080 Absolute Object.

RECORD TYPE	RECORD LENGTH	RECORD CONTENTS	CHECK SUM
1 byte	2 byte	(record length - 1) bytes	1 byte

The following record types are relevant to PROMDRIVER operation:

THEADR RECORD.

RECORD TYPE	RECORD LENGTH	MODULE NAME LENGTH	MODULE NAME	CHECK SUM
80	LLLL	NL	N...N	CC



LHEADR RECORD.

RECORD TYPE	RECORD LENGTH	MODULE NAME	MODULE NAME LENGTH	CHECK SUM
82	LLLL	NL	N...N	CC

MODEND RECORD.

RECORD TYPE	RECORD LENGTH	MOD TYPE	CHECK SUM
8A	02 00	00	74<

PEDATA RECORD.

RECORD TYPE	RECORD LENGTH	FRAME NO	OFFSET BYTES	DATA DD...DD	CHECK SUM
84	LLLL	AAAA	0A	DD...DD	CC

PIDATA RECORD.

RECORD TYPE	RECORD LENGTH	FRAME NO	OFFSET	ITERATED DATA BLOCK	CHECK SUM
86	LLLL	AAAA	0A	DD...DD	CC

ITERATED DATA BLOCK.

REPEAT COUNT	BLOCK COUNT	CONTENT
--------------	-------------	---------

REPEAT COUNT is the number of times the CONTENT field is to be repeated.

If BLOCK COUNT is non zero then the CONTENT field is that number of ITERATED DATA BLOCKS.

If BLOCK COUNT is zero then the CONTENT field contains a DATA COUNT byte followed by that number of data bytes. Maximum iteration depth is 17.

Load address is obtained from (FRAME NO * 16)+ OFFSET.

OFFSET must be less than or equal to 0FH.

PROMDRIVER responds to PEDATA and PIDATA records, and requires that the file ends with a MODEND record. All other record types are ignored, except for illegal types which cause a file format error to be reported.

When creating object files PROMDRIVER generates:

- an LHEADR record
- a THEADR record
- a variable number of PEDATA records
- a MODEND record

The MODULE NAME generated is MQP_ELECTRONICS

Intel 8086 Absolute Object files are limited to 1 Mbyte in size.

Intel 286 Absolute Object

This file format comprises the following sequence:

- a FILE HEADER (1 byte)
- a BOOTLOADABLE MODULE HEADER (95 bytes)
- a variable number of ABSTXT fields (variable length)
- a CHECKSUM (1 byte)

FILE HEADER.

One byte with the value A2H.

BOOTLOADABLE MODULE HEADER

TOTAL SPACE	RESERVED	ABSTXT LOCATION	RESERVED	LAST LOCATION	RESERVED
4 bytes	71 zero bytes	4 bytes	4 zero bytes	4 bytes	8 zero bytes

TOTAL SPACE is a 4 byte value indicating the minimum number of bytes in main memory needed to load the module.

ABSTXT LOCATION is a 4 byte value representing the offset in bytes of the ABSTXT section from the start of the file.

LAST LOCATION is a 4 byte value representing the offset in bytes of the last byte in the file from the start of the file.

ABSTXT field.

REAL ADDRESS	LENGTH	TEXT
3 bytes	2 bytes	LENGTH bytes

CHECKSUM is a single byte which is the complement mod 256 of all the previous bytes in the file.

Intel 286 Absolute Object files are limited to 16 Mbytes in size.

Appendix D

Warnings

Do not remove a device from the socket while the active LED is showing operation in progress.

It is not good practice to switch on or off the programmer with a device in the socket. This can damage some devices.

Warning on AUTO Typesetting:

Automatic type setting depends on the device in question having been provided by the manufacturer with a so-called Silicon Signature. Not all devices can be identified in this way. Indeed some devices can even be damaged by the special voltages involved. In particular some microprocessors which are claimed by their manufacturer to program like a particular EPROM using a pin converter will probably never contain the silicon signature. NEVER use auto type selection if you know that the device does not contain a silicon signature.

Avoid touching the pins of the device you are trying to read or program. Especially with surface mount devices, we highly recommend using a vacuum handling tool. This has the added advantage of lessening the chances of damaging the fragile pins of surface mount devices and their package converter sockets.

Use anti-static handling precautions at all times. An Earth Bonding Point is provided on your Pin-Master 48 to allow the connection of wrist straps conforming to EN100 015-1. Intermittent faults may indicate worn or damaged sockets, especially in surface mount package converters. Even with careful handling, sockets have a finite lifetime and may need to be replaced from time to time. Particularly in a dirty environment, we recommend covering the programmer when not in use.

If you ever need to transport the unit, please observe the following precautions:

- Pack the programmer carefully. If you cannot find the original purpose made packaging, please ensure that each item is well padded.
- The mains lead should be packed separately to avoid damage to the equipment from plugs.

Appendix E

Frequently Asked Questions

Q. Although devices blank check OK, they always fail to program, usually with a failure at a low address.

A. The devices are damaged or, especially for devices like the 87(C)51, one or more of the lock bits have been blown.

Q. I want to program a microprocessor or micro-controller that the manufacturer says programs exactly like an EPROM. How do I program it?

A. First, set the device type to the EPROM: note, you will not be able to use auto type select to do this. If large and complicated package converters are being used, you may have to reduce the access time for reading and checksumming the device. Also you will need to turn ID Check off. Then use Append Programming to program only the part of the EPROM's address space that exists in the micro-controller.

Q. I get a different checksum with this programmer than another manufacturer's programmer.

A. This programmer calculates the checksum by adding the contents of all bytes in the device. The bytes that are not defined in the programmed file are filled with the blank value for the device, usually FF hex. Some

other programmer manufacturers fill the unused parts of their devices, especially 8751s, with 00. To make the checksums the same, we have added a special feature to allow you to select 00 as the blank value when filling unused parts of devices.

Q. When I use auto device selection of a certain device why do I see the message:

Check Support Contract

A. This device was added to the menu after the expiry of software support for the programmer, so is not available without a new registration code.

Q. How do I know which sectors are locked on AMD 29F400B or 29F400T (and similar) devices?

A. Select the correct device type and display the device. The 11 word locations following the end address of the device (40000 hex onwards) will contain in ascending order the setting for each sector. If the least significant bit is 0 the sector is locked, if set to 1 the sector is not locked. For other devices in this family, check with your AMD data book to check how many sectors can be locked and where the end address of the device is located.

Q. After programming a 29F100B (or similar bootblock FLASH device) I locked one of the sectors. According to the programmer this operation was successful. However I still appear to be able to erase and program this sector.

A. The programmer uses a special hardware method to bypass the locking

mechanism. When this device is used in the target system it will indeed be locked.

Q. How can I find out which special features are enabled on Microchip PIC devices?

A. Select the correct device type, put the PIC device in the socket. The special features - configuration word, ID value and locked status can be seen by selecting Special.