BP Microsystems, Inc.
The Engineer's Programmer
User's Guide

COPYRIGHT AND LEGAL DISCLAIMER

5-Oct-99

BP Microsystems, Inc. *The Engineer's Programmer* User's Guide. BP DOS software version 3.xx (this number is subject to change as new software updates are available every 6 weeks).

BP Microsystems, Inc.
1000 N. Post Oak Blvd.
Suite 225
Houston, TX 77055-7237
PHONE: 800-225-2102, 713-688-4600 (outside US)
FAX: 713-461-7413
BBS: 713-461-4958
www.bpmicro.com

The information in this manual is subject to change without notice and, except in the warranty, does not represent a commitment on the part of BP Microsystems, Inc. BP Microsystems, Inc. believes the information in this manual to be correct at the time of publication; however, BP Microsystems, Inc. cannot be held liable for any mistakes in this manual and reserves the right to make changes to the product and product resources in order to make improvements.

Any mention of third-party products is for reference only, and does not constitute a recommendation or endorsement of these products.

TABLE OF CONTENTS

TABLE OF FIGURES

TABLE OF TABLES

CHAPTER 1
INTRODUCTION

Congratulations for selecting a BP Microsystems programmer - designed to perform the most demanding programming tasks. BP Microsystems programmers are used for design engineering, production programming, device testing, and field service. Your programmer's flexible hardware will continue to program the newest programmable devices by obtaining updates from BP Microsystems. BP Microsystems stands behind the programmers it manufactures to ensure your continued satisfaction.

BP Microsystems is committed to providing the worldwide market for device programmers with high performance innovative programmers and great customer support at the lowest price possible. Our programming platforms span the spectrum from low-cost single-family programmers to fully universal programmers that support virtually every device available today.

Every programmer built by BP Microsystems is built to the highest standards. Each programmer offers a wide range of device support, flexible pin drivers, protection from damage caused by operator errors and defective devices, stable hardware that does not require periodic maintenance or recalibration, and software that can be easily updated in the field to support the newest programmable devices. All of our programmers attach easily to the PC for ease of use.

The menu-driven software provides an efficient, user-friendly interface, and includes a full screen editor to view and modify your data, macro record/play facility for batch-file execution, and virtual memory management to deal with very large files. The programmers that support memory devices will accept all popular file formats, use intelligent identifiers to auto-select proper programming algorithms, provide selectable range programming on many chips, and have a special set programming mode for automatic collation of data. The programmers that support PLDs accept JEDEC files with editing ability for both fuse data and test vectors. The PLD programmers also support Altera's compressed **P**rogrammer **O**bject **F**ile (POF) format used with their MAX family of devices.

The programmer easily connects to the parallel printer port of any IBM PC or compatible from 8086 to 80486. The interface software operates in as little as 640K RAM standard on most computers but takes advantage of expanded memory, if present.

The **BP-1200** leads the competition in devices supported, programming yield, performance, ease of use, and cost of ownership. The ability to program almost every programmable device, including the fastest and largest programmable logic devices (PLDs), memories, and microcontrollers available, gives you the freedom to choose the optimum device for new designs. The BP-1200 can be field upgraded to meet your future requirements through expandable hardware that can support all package types (DIP, PLCC, SOIC, TSOP, QFP, and PGA) with up to 240 pins.

If you only use memory devices, the **EP-1140** is a high-performance low cost 40-pin programmer supporting most EPROMs available today. The EP-1140 also supports a variety of microcontrollers. The **EP-1132** programmer offers identical performance and near-identical device support; its 32-pin programming socket precludes 40-pin devices.

The **PLD-1128** programs almost any 20, 24, or 28 pin PLD currently available. The **CP-1128** offers even greater flexibility by also supporting many E/EPROMs and bipolar PROMs through 28 pins. The **PLD-1100** is the predecessor to the PLD-1128 and supports 20 and 24 pin PLDs. Even though the PLD-1100 is no longer produced by BP Microsystems, it is still supported and new devices are still being added to its support list at no cost to the customer.

The information in this manual is subject to change without notice and, except for the warranty, does not represent a commitment on the part of BP Microsystems, Inc. BP Microsystems, Inc. believes the information in this manual to be correct at the time of publication; however, BP Microsystems, Inc. cannot be held liable for any mistakes in this manual and reserves the right to make changes to the product in order to make improvements.

EPROM AND PLD PROGRAMMERS

BP Microsystems parallel interfaced programmers are software configured and controlled by a personal computer (PC). The software running on the PC provides the user interface, data buffer, algorithm data base, and control functions. Specific chip programming algorithms and instructions are stored on the PC's disk and downloaded to the programmer when you program a chip. Thus, the algorithm is actually executed by the programmer's internal microprocessor. This guarantees accurate voltages and proper wave forms independent of the PC's speed. The speed of your PC will only affect the rate at which the algorithm and data is downloaded to the programmer and will not affect programming yield. The programmer is upgradable through new software versions available to the end-user. The most recent software is generally the only requirement for the latest chip support (provided that your hardware has the proper number of pin drivers and the correct socket module). Using a PC for control creates several positive attributes: the programmer costs less, it is more convenient to update, it is easier to use, and is physically smaller than competitive programmers on the market.

BP Microsystems programmers are capable of programming and reading virtually every programmable device made. They program chips from data supplied by your

computer or another chip; they also let you read chips and store the data on your computer disk.

EPROM BACKGROUND

This section provides an elementary education on memory chips. The EPROM (Erasable Programmable Read Only Memory) is a nonvolatile memory that is erasable by exposing the silicon memory to ultraviolet light. The EPROM memory is located directly underneath a quartz window located atop the chip. An EPROM is known as *non-volatile* memory, meaning that it does not require power to retain its data and retains data programmed into it until it is erased. The EEPROM (Electrically Erasable Programmable Read Only Memory) is a non-volatile memory that can be erased and programmed electrically. Most EEPROMs can be erased and programmed a byte at a time. Flash EEPROMs are less expensive than standard EEPROMs, but you must erase the whole chip or a specific sector prior to programming. ROMs (Read Only Memories) are not user programmable; they are factory-programmed using a mask pattern when the chip is manufactured. PROMs are one-time Programmable Read Only Memories and usually employ a bipolar fuse-link technology. Production EPROMs are packaged in inexpensive plastic packages with no quartz window, so they cannot be erased. They are technically PROMs, but are usually called one-time programmable (OTP) EPROMs. Non-volatile Random Access Memories (NVRAMs) are typically static RAM chips (SRAMs) with internal battery back-up. The battery allows the chip to retain data for five to ten years and permits programming prior to board insertion. The term EPROM is often used as a general classification of the above mentioned non-volatile programmable memory technologies. Microcontrollers are microprocessor chips containing RAM, I/O, and some form of non-volatile memory. The non-volatile memory may be programmed to contain a user-defined program making the microcontroller perform a specific task.

A memory chip can be described by its organization: how many bits are read at one time (a word) and how many words are in the chip.

> ***Example:*** *A 27256 EPROM contains 32,768 eight-bit words, or 32K-bytes. A 27210 contains 65,536 sixteen-bit words. Most EPROMs come in two widths: byte-wide (8-bits) and word-wide (16-bit words). Multiplying the data width by the number of words gives the size of the array, in bits. The 27256 contains 262,144 bits (256K-bits); the 27210 contains 1,048,576 bits (1-megabit).*

Sets of EPROMs are easy to program thanks to the *Set* programming mode under the *Device/Options* command. It will prompt you to insert each chip of the set while programming or reading, automatically splitting data files for wide data paths or multiple banks of chips. When using 16-bit wide chips, the byte-order is configurable to conform to Motorola or Intel conventions (standard, or byte-reversed).

The data stored in a chip may represent various kinds of information: a program, a character generator, a table of waveforms, a logic pattern, etc. Data patterns can be represented in your computer in many ways. The most straightforward method is a

binary file. It contains an exact copy of the data to be placed in the chip (a 27256 file would contain 32K-bytes). The difficulty with binary files is that they may contain characters that cannot be printed (only 95 ASCII characters print, while the file may have 256 different characters). The solution to this problem lies in hex files.

HEX FILES

Hex files represent the data in your chip with ordinary lines of text. Each data byte is converted into two "hex" characters (0-9, A-F), each representing four bits. Address, checksum, and other information may also be included in the file. BP Microsystems EPROM programmers support all popular hex file formats: straight hex, hex-space format, ASCII hex, Intel MCS-80, Intel MCS-86, Intel MCS-386, Motorola S(1-9), Tektronix Tekhex and extended Tekhex. These different formats offer the flexibility to communicate with almost any system generating data for EPROMs. Any file format can be used with the data from any chip. It doesn't matter which format you choose to store the data.

The control software includes a full screen editor capable of displaying any combination of binary, ASCII, hex, or octal representation of your data. You can use it to view or edit data, set file checksums, copy data, fill ranges, etc. The software supports binary files and all the hex file types mentioned above.

ASSEMBLERS AND COMPILERS

Many users will be writing a program, compiling, assembling, and programming the resulting object code into an EPROM. It is necessary to get your data into a format compatible with the programmer since the interface software does not, and cannot, support the hundreds of intermediate file formats used by assemblers and compilers. You need to compile, assemble, link, and load your program. The output of the loader should be in hex or binary. Linkers and loaders producing other formats usually come with a utility program that will convert the output to Intel hex or some other common format.

PLD BACKGROUND

Programmable logic devices (PLDs) are logic chips that have their internal logic configuration determined by the user, not the manufacturer. The design engineer customizes the parts for specific applications. Using PLDs can reduce parts count, increase reliability, increase performance, reduce design time, and reduce manufacturing costs—all at the same time!

The term PLD can be applied to all varieties of programmable logic chips including erasable programmable logic devices (EPLDs), PALs, PLAs, GALs, PEELs, FPLs, sequencers, PROMs, and a slew of other parts. The various acronyms are trademarks for families of related parts.

PLDs are programmed by placing them in a special mode where they are accessed like memory devices. The programmer then stores a fuse pattern written by an engineer (or read from another chip) into the logic array of the PLD. When the PLD is returned to a normal operating mode, it performs the customized logic function programmed into it.

Programmable arrays used in PLDs are based on the same technology used for today's memory chips. The fastest parts employ bipolar logic and fusible metal links which melt during programming. CMOS technology allows PLDs to be built using EPROM and EEPROM techniques, where electrically charged cells are used to represent fuse states. Naturally, these parts can be erased (by UV-light or by the programmer) and reprogrammed with a new pattern.

LOGIC COMPILERS

Software is available to help the engineer develop designs using PLDs. Tools called logic compilers perform the tedious task of translating a design file written in a high-level language into a fuse pattern stored in a standard JEDEC (Joint Electronic Device Engineering Council) file. The input to the compiler, usually a high-level language or graphical schematic diagram, varies widely depending on which product you are using, but fortunately the output file format is standardized. JEDEC files are produced by almost all PLD development software and are accepted by all popular PLD programmers. Some compilers use the POF format, which is also supported.

There are many commercial software packages available to help you design with PLDs. There are tools available from semiconductor manufacturers very inexpensively that support only a single brand of devices, as well as universal development software available from third parties. PALASM is a program available from Advanced Micro Devices for a nominal charge and supports virtually the full line of AMD and MMI devices. It is probably the most flexible of the inexpensive software packages. Other single-family inexpensive packages are available from National Semiconductor, Signetics, and Texas Instruments.

MINC, Inc. offers a universal development product system with comprehensive features and device support. It offers a variety input formats and advanced features such as automatic device selection and partitioning. Other universal products include ABEL from Data I/O, and CUPL from Logical Devices.

ABOUT THIS MANUAL

This manual has been built in such a way as to support any one of the following products using the DOS-based BP Control Software.

- BP-1400
- BP-1200
- EP-1140
- EP-1132
- PLD-1100

- PLD-1128
- CP-1128

If this is your first experience with a BP Microsystems programmer, you should definitely take advantage of *Chapter 2, Getting Started* and *Chapter 3, Running a Job from Start to Finish*. These chapters will take you step by step through the setup, installation and programming processes. Along the way, you'll get acquainted with the full range of the programmer's capabilities.

If you are already familiar with our programmers, you may want to use the *Appendix B, Quick Start Guide* located at the back of the manual, which covers the key points of getting the programmer up and running. We still encourage you to explore the manual for useful tips on getting the most from your BP Microsystems' programmer.

- Chapter 1 introduces you to and familiarizes you with the Engineer's programmer, covering the basic definition of a programmer, along with some other useful industry information.
- Chapter 2 is a getting started process that takes you step-by-step through plugging in, turning on and setting up the programmer for programming.
- Chapter 3 is an in-depth look at programming from start to finish.
- Chapter 4 covers descriptions and usage of the data editors.
- Chapter 5 describes the different file formats that are used in programming devices.
- Chapter 6 is organized by software function and command, listing details and usage about each.
- Chapter 7 covers hints, tips and other useful information.
- Chapter 8 addresses using macros.
- Chapter 9 explains the additional software, JobMaster, and instructs usage.
- Chapter 10 covers the basics of serialization of programmed parts.
- Chapter 11 defines test vectors and gives examples.
- Chapter 12 lists the procedure for erasing EPROMs
- Chapter 13 covers emulation modes.
- Chapter 14 covers error messages that are not self-explanatory and helps you identify problems and mistakes.

This manual also includes a glossary of common terms and acronyms as well as an extensive index at the end of the manual.

Certain character formats have been used to allow us to assist you, the reader, with messages and instructions. Below is a list of the codes that pertain to this manual.

- Warnings are set off by a lightning bolt (↯) icon and indented, **bolded** text.
- Notes are set off by a notepad (📝) icon and indented, *italicized* text.
- Examples are set off by the bolded word "**Example"** and indented, *italicized* text.

Action words within the manual are set off as listed below:

- **<Bold>** – actual keystrokes on the keyboard, i.e. press <**Esc**>
- **Bold** – important text found within a paragraph or statement and options found within the software, i.e. **YES**.
- ***Bold Italics*** – actual phrases found within the software, i.e. ***Device Selector:***.
- *Italics* – software fields and drop-down menus, i.e. *Operator Mode* or *JobMaster/Configure.*

Exceptions: If a keystroke is listed in a note or example, the text will be bolded but not italicized. If the item is found in a warning, it will not be bolded.
If a software field or drop-down menu is found in a note or example, the title will be regular text (without format). If the title is found in a warning, it will be italicized but not bolded.

Any page number referenced within an on-line manual is a hotspot and therefore can be clicked on to be taken directly to that page.

CHAPTER 2
GETTING STARTED

THE BASICS

If you are completely new to device programmers, there are a few basics you should know before continuing through the manual.

- A device programmer is a tool used to configure a programmable integrated circuit for use in the design or manufacture of electronic equipment. The blank programmable devices are available from many manufacturers for many different applications.

- The device programmer is analogous to a floppy disk drive. A floppy drive allows a user to copy programs or data stored on a PC onto a blank floppy disk. The programmer copies programs or data from the PC to a blank integrated circuit instead of a disk.

- The user may design a pattern manually by using an editor, or may use a compiler. Compilers, available from third parties, will generate software to be programmed into memory devices or microcontrollers, or logic functions to be programmed into Programmable Logic Devices (PLDs).

- The BP software supplies the programmer with all the information it needs to program the user's pattern into a specific device. The software is updated eight times a year to cover new parts and to update existing algorithms. Learn more about the individual programmers found in this series and unique capabilities for each in *Chapter 1 – Introduction.*

INSTALLATION

HARDWARE INSTALLATION

After the equipment has been unpacked and inventory of the boxes has been confirmed (refer to *Appendix C, Operator Checklist*), you are ready to connect the programmer to the PC. Your new programmer will be connected to a parallel printer port on your PC. It is preferable to dedicate a port to the programmer. You may plug and unplug the cables or use a manual mechanical printer switch instead.

- Use the 25-conductor cable provided with your programmer.

  ✍ **Do not use a ribbon cable or an RS-232 cable that has fewer than 25 conductors. You can extend the cable up to 12 feet, but be certain to use only a 25-conductor shielded cable (available from BP Microsystems, Inc). A high percentage of the hardware-related cable failures reported to our technical support are actually ribbon cable failures. Ribbon cables work well installed inside a chassis, but often make poor connections when subjected to the flexing that occurs when used improperly.**

  ✍ **Set the line voltage selector on the PC to the appropriate position to avoid damage!**

- Plug the programmer AC power cord into a power socket.

☐ *The programmer power supply operates from 90 to 250 VAC for simplified worldwide use. Connect one end of the DB-25 cable provided to the programmer's connector and tighten the screws. Connect the other end of the cable to your computer's parallel printer port.*

  ✍ **Do not attempt to use any print buffers, electronic switches, or software copy protection keys on the same port as the programmer. Verify that you have connected to the correct parallel port on your computer. Connecting to a serial port or a third party card may damage the programmer. This type of damage is not covered by the warranty.**

- Turn on the computer and the programmer. Both the Power LED and Active LED on each programmer site will light up. While the Active LED is on, the programmer is performing a Self-Test. After several minutes, the Active LED will turn off and only the Power LED will remain on. If any of the Fail LEDs have turned on, the programmer has detected an error during its Self-Test. If this occurs, make a notation of which unit is displaying the Fail LED and call BP Microsystems technical support line.

POSITIONING THE COMPONENTS

Proper placement of the system components is an important factor in the ease and efficiency with which you can program devices.

You should position the programming modules and PC keyboard so that all can be reached easily from one working position. You should keep the programmer level if you intend to program fine-pitch devices.

We also recommend setting up on a conductive mat that can be grounded. The operator should use an ESD wristband plugged into the programmer to prevent static discharge that can cause damage to the devices you are programming.

Be sure to allow extra space for staging blank and programmed devices, labeling and paperwork. We suggest designating an area to your left for blanks, an area to your right for programmed devices, and a location to the left rear for rejects. After time, you will develop a habit of moving in the same pattern, thereby increasing your efficiency.

CONNECT THE COMPONENTS

Once you have things in place and the software installed, you are ready to connect the components of the system.

1. If you have more than one programmer and only one PC, daisy chain the programmers to each other using the cables provided.

2. Plug the power cords from each programmer and the PC into a grounded power socket.

3. Now you can connect the programmer to the PC using the 25-conductor data cable provided with your system. Do NOT use a ribbon cable or other substitute as this may cause system problems later.

4. Connect the data cable to one of the parallel printer ports on the PC. Connecting to a serial port or a third-party card may damage the programmer.

It's most efficient to use one port for the programmer and a different one for your printer. If we supplied the PC as part of your system, you should attach the data cable to LPT-1.

5. Connect your ESD wristband to the grounding outlet on the front of the programmer. If you are working on a grounded mat, connect the mat to the programmer and the ESD band to the mat.

SOFTWARE INSTALLATION

If you have purchased a PC with your programmer system, the software will already be installed in the C:\BP directory.

If you are installing the BP software on a PC not provided by BP Microsystems or another computer, use the following instructions:

1. At the DOS prompt, type in the drive and directory where you would like the BP software to be installed. Confirm that this destination is now the visible prompt.

2. Insert the disk labeled **Disk 1** into Drive A and type **a:install** at the prompt line. Follow the instructions for installing the disk contents and inserting the two remaining diskettes. This will extract the BP.EXE software and JobMaster.DLL onto your selected directory.

3. Verify that the programmer is powered on and all Pass LEDs are on.

4. Type <**BP**> at the command prompt and press <**Enter**>. The software should display a screen similar to this:

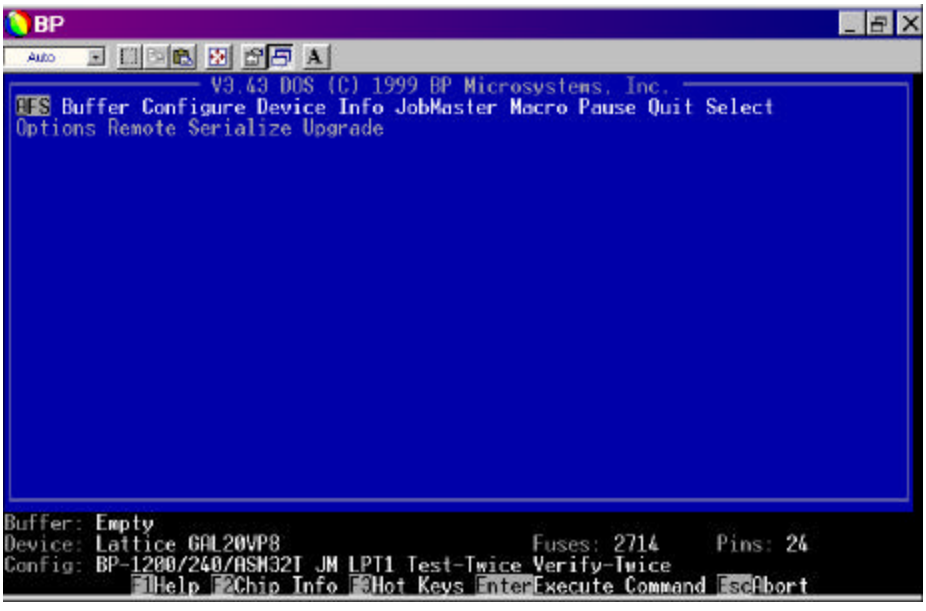


Figure 1 – Initial Sign-on Screen

The software version number will appear at the top of the screen and a message will appear briefly at the bottom saying *"Establishing communications, please wait."* If the programmer is found on one of the LPT ports, no error message will appear and you are ready to begin programming.

If an error message prompts a problem, you should make sure you are connected properly, the power to the programmer is turned on, then try again.

If the software does not detect a programmer, or if no programmer is attached, it will automatically go into a demonstration mode and allow you to use all the features that do not require a programmer to be present.

☐ *When powering on the system, always turn on the programmer before launching the software.*

Programmer Self-Test

Each programmer will automatically perform a Power On Self Test (POST) every time you turn it on.

Both the Power LED and Active LED on each programmer site will light up. The Active LED will stay on until the POST is completed. The Power LED will remain on. If any of the Fail LEDs have turned on, the programmer has detected an error during its Self-Test. If this occurs, make a notation of which unit is displaying the Fail LED, which lights are steady and any that are flashing, then call BP Microsystems technical

support line (contact numbers for Technical Support are listed in *Chapter 14, Troubleshooting and Maintenance*).

We recommend running a full system Self-Test on your programmer weekly, but first you'll need to have the BP software up and running.

RUNNING SELF TEST

Even though the programmer does an initial Self-Test upon start-up, you are strongly advised to run a full system Self-Test on your programmer before performing any operations. The BP Microsystems diagnostics will ensure that the power supplies are functioning properly, and will test the integrity of all pin drivers shipped with your particular configuration. The diagnostic Self-Test is activated by pressing <**Alt-D**>. Insure that there are no devices in the programmer sites prior to beginning the Self-Test. Any device left in a programmer site may be damaged during testing.

Once the Self-Test has been entered, two choices under the *Device* main menu selection will be visible. Highlight **Test** and press <**Enter**>. You will be given a choice between running all the units through Self-Test or selecting a particular unit to test. Choose **All** and press <**Enter**>. The Self-Test will cycle through all of the units on the programmer.

When a unit finishes, the green Pass LED will be on. If any Fail LEDs are illuminated after the completion of the Self-Test, note which unit(s) have failed and call BP Microsystems technical support.

SETTING UP THE SYSTEM

Check the startup screen to verify that the model number of your programmer and port (LPT-1) appear on the "Config:" status line. If the word "DEMO" appears, check your connections to the programmer and make sure power is on. Restart the software if necessary.

In *Chapter 2, Getting Started*, you learned about the components that make up the programmer and how to set up the system.

In this chapter you learn, as a first-time user of a BP Microsystems' programmer, step-by-step, the basics of efficient operation. Our approach is a little different from other systems with which you may be familiar, so please take the time to go through each chapter.

If you are already familiar with a given section, feel free to skip to the next. But even if you have already used BP programmers, this manual will give you some special tips for using the system more effectively.

BP Microsystems has designed each programmer to be as simple to operate as possible, and we think you'll find it a pleasure to see how quickly you can learn to use the many features offered.

*Don't forget, you can always get context-sensitive Help by pressing the <**F1**> function key.*

NAVIGATING THE SYSTEM

The initial software screen is known as the *Command Mode* screen. From here, it's easy to navigate your way around the system, using the menu commands. You'll see the menu commands on the line near the top of the display, beginning with *Buffer* on the left.

The BP system offers a wide variety of commands and options, which are covered in detail in *Chapter 6, Command Reference*. The commands commonly used most often will be introduced in this chapter.

When you're in Command Mode, you can access any of the main menus by typing the first letter of the menu name. You may also press the <**Left**> and <**Right**> direction keys to change the selection, then press <**Enter**> to execute the selected command.

Most of the main menu items have a sub-menu of further options. To move through these options, use the <**Up**> and <**Down**> arrow keys to highlight the desired selection, then press <**Enter**>.

You can return to the *Command Mode* screen by pressing <**Esc**>. In the instance that you are in a sub-menu, you will have to press <**Esc**> twice.

Many different "hot keys" are available in *Command Mode.* These key combinations are short cuts for more time-consuming menu selection techniques. Learning to use them can speed your operation of the system. All the hot keys are listed in detail in *Chapter 6, Command Reference*. You can see them on-screen at any time by pressing <**F3**>. The most commonly used hot-keys will be introduced within this chapter.

FULL SYSTEM SELF-TESTS

Now that the system is up and running, press <**Alt-D**> to perform a full system Self-Test. This will test every component in the system more extensively than the automatic POST. We recommend that you perform this test once a week, to make sure everything is operating properly.

NEED HELP?

The most important "hot key" when you're getting started is <**F1**>. You can press the <**F1**> key for extensive help at any time.

CONFIGURING THE SYSTEM

You will learn how to customize the system to serve your particular needs, using options under the *Configure* menu. The system "wakes up" with default options, which should

serve your needs initially, but the default settings can be changed to customize the programming process to better suit your needs.

READING THE SYSTEM STATUS

The four status lines at the bottom of the *Command Screen* display important information to help you double-check what you've told the system to do. These lines will show you:

- What data is in the buffer (there should be none now, because you have not yet loaded a file).
- The device selected and its size information (again, there will be none listed until you select a device in *Chapter 3*).
- The programmer attached to your PC (this should indicate a programmer as well as the type and number of programmer sites) and the *Device/Options* selections you have made.
- General status. This may tell you to press a key to continue, or to define what the function keys do.

From the *Command Mode* screen, use the cursor keys to select *Configure*, and press <**Enter**>, or simply type <**C**>.

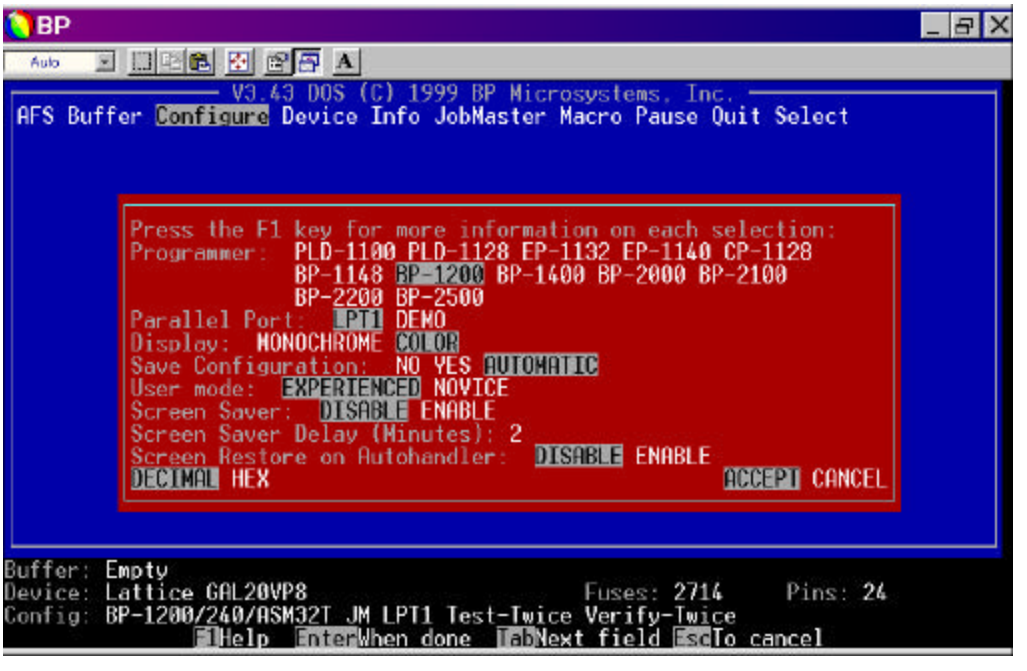


Figure 2 – Configure Options

CONFIGURE OPTIONS

Your BP software starts up with certain default options set. Even though these default options are set, you can change them to customize the system for your particular needs. These changes can also be made permanent, as discussed further within the manual.

Programmer

The software will automatically recognize what type of programmer you have attached to your PC. Make sure that it matches your programmer. You can change this if you wish to see the different device support options for another programmer.

Parallel Port

This field shows the parallel port to which the programmer should be attached. If the programmer is attached to a different port, the software will find it automatically. If no programmer is attached, the software will put itself into a demonstration mode.

Display

This field allows you to switch from color to black-and-white display. You may find this useful if you're using a color laptop.

User Mode

This field tells the software how much on-screen help you need. The default option is ***NOVICE***. This setting provides you with numerous warning messages to guide you in learning your way around the system. It also gives you the chance to abort potentially hazardous operations, like programming a chip. By changing the setting to ***EXPERT***, you can turn off these messages when you no longer need them. This will make programming go a little faster.

Startup Messages

You can set this field to ***DISABLED*** to turn off the startup messages, again speeding up your programming routine.

Screen Saver

This field will enable a screen saver that will blank your screen after a specified amount of time.

Screen Saver Delay

Specifies the time of inactivity (in minutes) to wait before blanking the screen.

Screen Restore on Autohandler

This option will keep the screen from being blanked as long as there is ongoing communication with an autohandler.

Decimal or Hex

This field determines the format you use to enter numbers into the system. If you're used to decimal (0-9), select that option. If you use hexadecimal, (0-9, A-F), select **Hex**. The advantage of Hex is that it allows you to use fewer characters to represent bigger numbers, e.g., "FF" instead of "255". This is a global command, which means that changing it here will change it throughout the program.

SAVING YOUR CONFIGURATION

Whenever you want to change the Configuration of your system, you can make the BP software "wake up" to your settings every time. To do this, use the *Save Configuration* field.

The three options in this field are ***YES***, ***NO*** and ***AUTOMATIC***.

Yes

YES saves your choices when you press <**Enter**>. The next time you start the program these settings will be recalled. Your settings are saved in a .CFG file in the same directory as the BP.EXE file. If you want to revert to the original default settings, just delete the .CFG file.

No

Selecting ***NO*** lets you make changes to the system for the current programming session. When you start the program again, it will revert to the previous, or default, settings.

Automatic

The ***AUTOMATIC*** option saves any changes you make automatically whenever you quit the program. When you start the program again, all your settings will be restored, including your chip selection, saving you time in getting back to work.

MULTIPLE CONFIGURATIONS

You can keep several frequently used configurations on disk by saving them to different directories. You can override this feature by placing a line in your AUTOEXEC.BAT file similar to this:

set bpcfg = c:\bp

CHAPTER 3
PROGRAMMING FROM START TO FINISH

This chapter will take you through the entire process of starting and completing a programming cycle. The following sections are designed to take you through the step by step process of choosing the correct devices, loading and storing in the data buffer, and monitoring the entire job.

SELECTING A DEVICE

Before you can begin programming, you must tell the software which device you will be programming. The BP software supports thousands of devices, but it's up to you to designate which one and what type.

- **It is essential to choose the correct entry for the device you want to program. Programming algorithms vary widely between different semiconductor manufacturers and even between parts from one manufacturer with different speed ratings! Selecting the wrong algorithm can destroy your chip! The part number on your chip may have package code and temperature rating letters following the part number shown in the menu. When you select a chip, it is a good idea to specify both the manufacturer and the part number of the device you want to program. You must be careful to select the part number from the menu list that exactly or most closely matches your device part number, including the suffix that represents programming voltage and speed.**

- **Almost all programmable memory devices manufactured today support an electronic code that can be read to identify the correct programming algorithm. Most likely, your chip supports this feature and the code will be read before programming to ensure that the correct algorithm is being used. However, if you are in possession of a part that does not utilize this electronic code, selecting the correct chip from the menu is imperative.**

SELECT OPTION

From the *Command Mode* screen, type <**Alt-S**> or use the directional arrow keys on the keyboard to choose the *Select* menu.

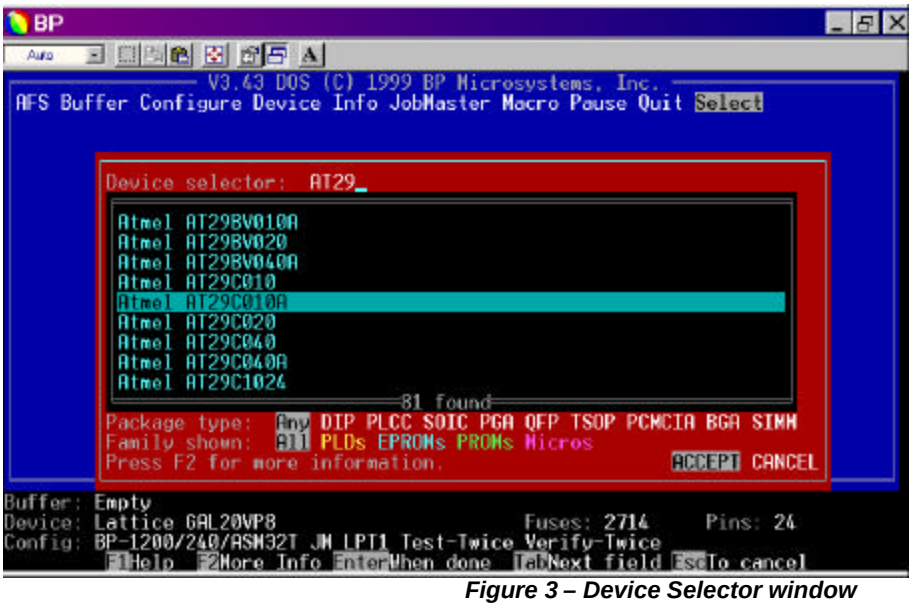


Figure 3 – Device Selector window

The *Device Selector* dialog box contains a very long list of devices. You can use the arrow keys to scroll through the list or you can utilize the search engine function.

To use the search engine, use the directional arrow keys on your keyboard to place the cursor next to the phrase "***Device Selector***:". Begin typing in the device information. As you type in the manufacturer's name and device number, the device list narrows to include only those names containing the characters you type, in the order you type them. Press <**Enter**> to select the device you want to program. The software should know the proper algorithm for programming it.

- *You can speed the search somewhat by using the* Package Type *and* Family Shown *fields to narrow the selection to DIP packages, for example, or PLD's. But if you know the manufacturer name and device number, the quickest way is to leave* Package Type *set to* Any *and* Family Shown *set to* All *and just type in the full device name.*

SAVING THE SELECTION

If you have set *Configure/Save* to **AUTOMATIC** or **YES**, the software will "wake up" with the same chip selected the next time you start it. If you have not set the *Configure/Save* option and would like to, refer to the previous chapter for assistance in doing so.

LOADING THE BUFFER

Now that you've selected a device, the programmer knows the proper algorithm (i.e., the correct voltages, pin numbers and other mechanical data). The next step is to load a file into the programmer's buffer.

UNDERSTANDING THE FILE FORMATS

A programming file may come "off the shelf" for common applications, or it may be custom-created for your company's particular application.

Choose *Buffer/Load* from the main menu. To limit the list to a specific set of files, you can change the "*Directory:*" specification from ***.*** to ***.hex***, for instance (press <**End**>, <**Backspace**>, **then type H E X**). You can also change the disk drive and directory by editing the "*Directory:*" field. To choose one of the files listed, select the file by moving the cursor up or down with the directional arrow keys and press <**Enter**> to copy your choice to the "*File to load:*" field and the software will automatically identify the file type and move its cursor accordingly.

☐ *Pressing <**Enter**> on "..\" or any other name ending in "\" is an alternative way to change the current directory and allows easy traversal through your directory tree.*

One of the big advantages of the BP software is its ability to recognize the type of file you have loaded, instead of requiring you to specify it. In the rare instance where the file type is incorrectly identified, you can change it in the *File Type* field.

For most work, you only need to know the correct name for the file you are programming into your device. If you are interested in editing or modifying files, turn to *Chapter 4, Using Data Editors* and *Chapter 5, File Formats* for further instructions.

Device/Options

In some cases, you will need to specify some additional Load Parameters to control how and where the file is loaded into the buffer. You can access these options by pressing <**F8**> from within the *Buffer/Load* dialog box. When you use these options, you can find instructions in *Chapter 6, Command Reference* under *Buffer/Load*, *Device/Options* and *Device/Configure*.

LOADING THE FILES

From the command mode, press <**Alt-L**> to bring up the *Buffer/Load* screen. In the *Directory* field, type the name of the directory in which the file you need is located.

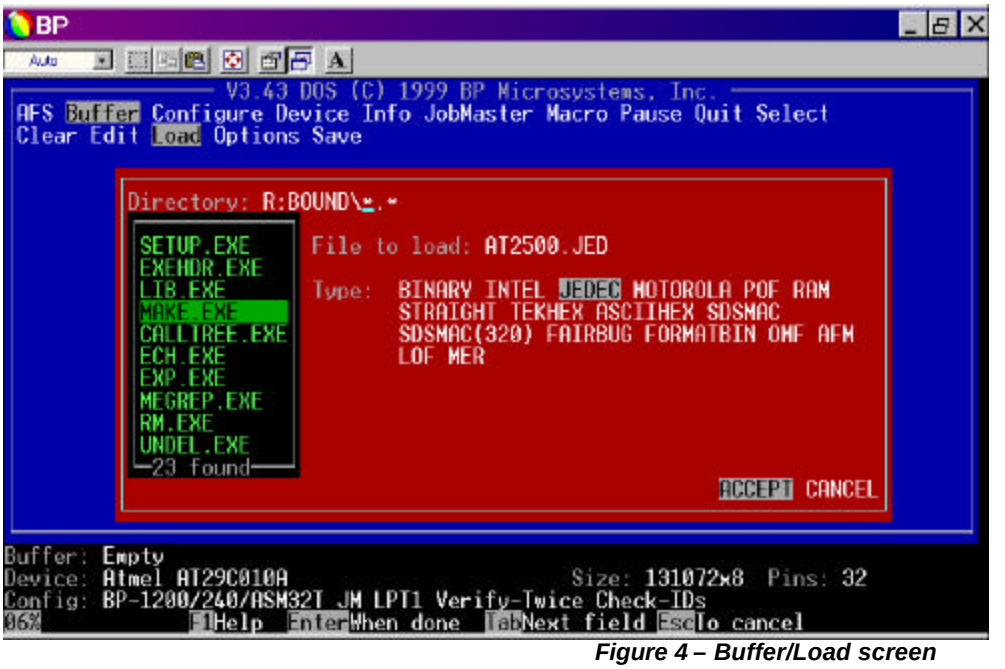


Figure 4 – Buffer/Load screen

If you are going to use the same file repeatedly, you can speed your operation by copying that file from the floppy disk into a directory on your hard drive. You can also set the *Buffer/Options* menu option to ***AUTOMATIC*** to find and load this same file every time you launch the BP software.

Use the cursor keys to select the file and press <**Enter**>. The file name will appear to the right of *File to load:* and the file *Type:* will be automatically determined. Press <**Enter**> again and the file will be loaded into the programmer's buffer.

You can confirm that this has been done by looking at the status lines at the bottom of the screen. The *Buffer:* field should have changed from ***Empty*** to the name of the file you selected.

PROGRAMMING A DEVICE

Now that you have a file loaded into the buffer, you are ready to begin programming a device.

SET NUMBER OF DEVICES TO PROGRAM

Before you get started, you must tell the software how many devices you want to program. Beginning at the *Command Mode* screen, select the *Device/Handler* menu and press <**Enter**>.

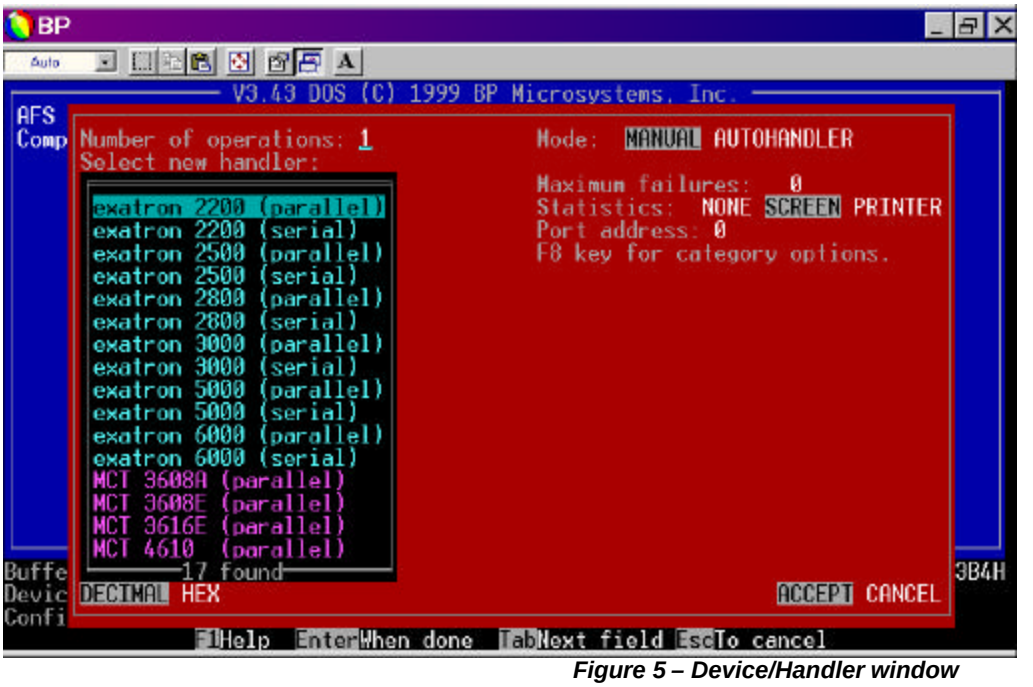


Figure 5 – Device/Handler window

In the *Device/Handler* dialog box, enter the number of devices you want to successfully program in the *Number of operations:* field.

- **We highly recommend using a vacuum pencil to pick up and insert a chip, in order to minimize the chance of damage from static discharge.**

- **Parts must *not* be inserted or removed when the ACTIVE LED is on.**

The programmer has protection circuitry, so it is not necessary to remove the device to be programmed before the power is turned on or off.

It is very important to insert the device correctly so that its pin 1 matches up with pin 1 in the programmer site. Pin 1 is often identified by a notch or a mark on the device, and by a mark on the programmer site. Inserting the chip improperly may damage it, so make sure you have it properly aligned.

CHIP PLACEMENT

DIP

The chip should be “top-justified”, which means pin 1 of the chip (the pin nearest an identifying dot or notch) goes in the top left corner of the socket.

SOIC and TSOP

Same as the DIP placement; pin 1 should be top justified.

PGA

Place the PGA device as indicated by the particular adapter or socket module.

QFP

Align pin one with the mark on the programmer site.

PLCC and LCC

Place pin one on the side facing the operator.

Once the device is in the programmer site, lock it in place to achieve continuity.

Now press the **Start** button next to that programmer site. The yellow ACTIVE LED will come on and programming will begin. When programming of that part is completed, the yellow ACTIVE LED will go off and one of the other two status lights will come on.

VERIFY THAT THE PART IS CORRECTLY PROGRAMMED

There are two ways to make sure the part is correctly programmed. You can watch the lights beside the programmer site. If the green PASS LED comes on, the part has been properly programmed. If the red FAIL LED starts flashing, something has gone wrong. You can get the same information by watching the PC screen, which will report the PASS/FAIL status of each part.

The worst mistake a device programmer can make is to allow a failed device to be counted as passed. The programmer has a special feature to help make sure this doesn't happen to you.

If the device has passed, the status LEDs will turn off and the **Start** button will turn on again when you remove the part from the programmer site. If the part has failed, the red FAIL LED will remain steadily illuminated even when the part is removed.

If you have to leave the programmer during a session, you can always determine the PASS/FAIL status of a part by looking at the LEDs. If a part is in the programmer site and the PASS LED is on, the part has been successfully programmed. If the part has already been removed and the FAIL LED is still on, that part has failed. If the part has been removed and only the **Start** button LED is on, that part was good and the programmer site is ready to program a new part.

As a result, the worst thing that could happen is that a good device will be programmed twice.

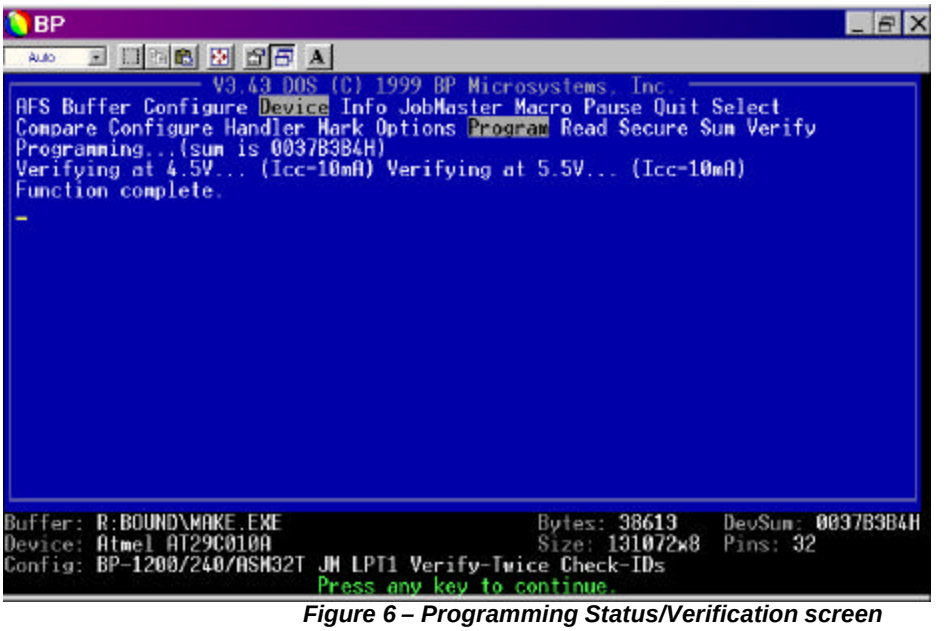


Figure 6 – Programming Status/Verification screen

You are now ready to program the remaining parts. Insert another device in the next programmer site, achieve continuity, and the system will automatically begin to program that part. Without waiting for it to finish, you can insert the next part into the next programmer site. By the time the last programmer site is filled, the first part should be completed, and you can remove it and insert a blank part in its place. Continue until all devices are done.

Once all parts have been successfully programmed, the PC will beep to indicate your production run is completed.

READING THE RESULTS OF YOUR PRODUCTION RUN

When the specified number of parts have been programmed, the software will send a report to the PC screen. This will tell you the device, the name of the file you programmed into it, how many operations the system performed, how many parts passed and how many failed. This is useful in helping you make sure you have programmed the correct number of parts successfully.

The report will also show you the beginning and ending time of your programming session, the number of units per hour, and the percentage of successful operations.

SAVING AND PRINTING THE REPORT RESULTS

The report from each run can be saved to the hard disk, and can be printed out. Instructions for doing this can be found in *Chapter 6, Command Reference* under the *Info/Log* command.

EXPLORING YOUR OPTIONS

In this section, you can explore some of the programmer's options for increasing your programming speed and productivity. You will also look at the system's flexibility in handling some unique programming challenges.

DEVICE SPECIFIC OPTIONS

Unlike other programmers, BP Microsystems' programmers do not append configuration parameters to data files. This allows the operator a greater degree of flexibility in programming devices. These options are chosen through the *Device/Configure* menu.

Device/Configure

The *Device/Configure* menu offers some very device-specific options, which allow you to customize your programming to meet specific requirements. Depending on which device you select, you will be presented with different options.

Some microcontrollers allow setting of such things as the memory mapping, watchdog timers, security, and clock source. Other devices have special operating modes that can be configured with this command.

Since *Device/Configure* options will vary from chip to chip, you should refer to the data book for your particular device for more information on the configuration parameters. You can always press <**F1**> for more help on each item in the dialog box.

Device/Encrypt

Some microcontrollers support a special level of security that encrypts your program and data. This allows you to protect your design secrets, yet still be able to verify the contents of devices after programming. More detailed instructions on encryption are found in *Chapter 6, Command Reference* under *Device/Encrypt.*

CHECKSUMS

Checksums are a useful way of verifying that the information you have programmed into a device is correct. There are two situations in which the checksums may appear to indicate an error when in fact you have programmed a device correctly.

The BP software calculates checksums based solely on the data portion of a file. Other programmers may calculate checksums differently, so you may see a different result than expected.

If you have edited a file and neglected to update the checksum, this too can indicate a programming error when in fact programming is successful.

For more details on checksums, see the *Buffer/Load* command in *Chapter 6, Command Reference,* or look up Checksums in the index.

CHAPTER 4
USING THE DATA EDITORS

The data editors give you access to memory data, fuse data, and test vectors throughout the *Buffer/Edit* and *Buffer/Vectors* commands. The *Buffer/Edit* command will bring up the ***Memory Data Editor***, if an EPROM or microcontroller is selected or if a non-JEDEC file has been loaded. It will bring up the ***Fuse Data Editor*** if a PLD is selected or a JEDEC file has been loaded. The *Buffer/Vectors* command appears only when a PLD is selected and will bring up the ***Test Vector Editor***.

MEMORY DATA EDITOR

The memory data editor allows you to look at data obtained from a binary or hex file, or from a memory chip. You have the option to edit the data and then store that data in a file. The edited data may also be used to program a chip.

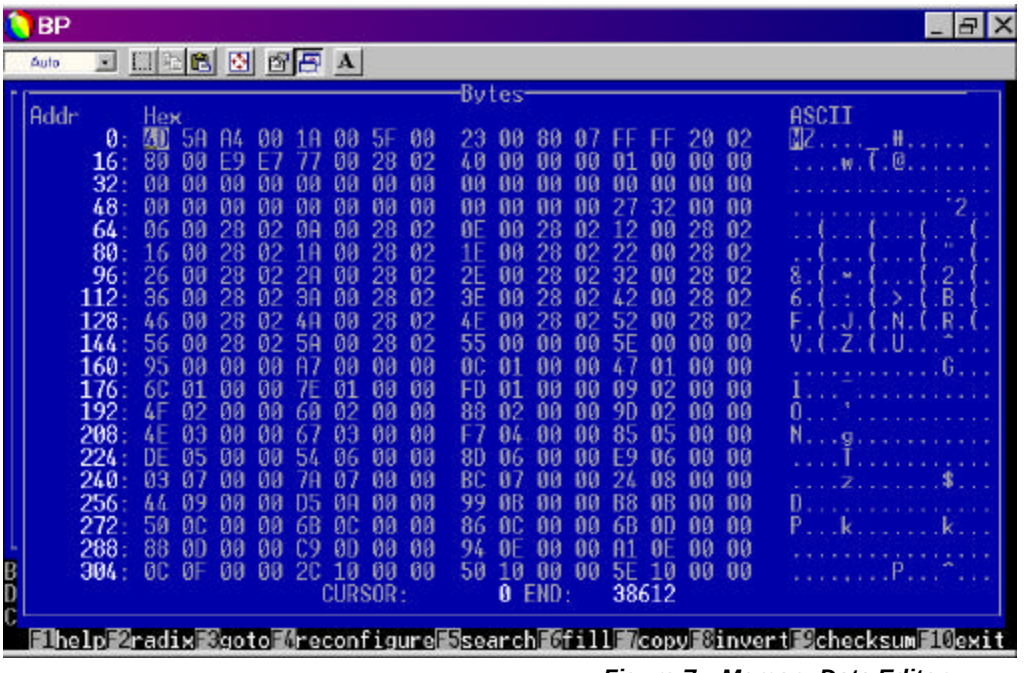


Figure 7 – Memory Data Editor

Using the keyboard commands listed below, you can access the data and alter the format it is presented in. The data is generally displayed in ASCII or hex formats, but you may change the data for binary and octal formats as needed.

KEYBOARD USAGE

¬, - ®, ¯ PgUp, PgDn	Move cursor
Ctrl-PgUp	Go to top of buffer
Ctrl-PgDn	Go to end of buffer
F1	Help on key usage
F2	Select address radix (decimal/hex)
F3	Set Cursor Address (goto)
F4	Reconfigure (hex, ASCII, octal, binary)
F5	Search for pattern
F6	Fill range with constant
F7	Copy a range of data
F8	Invert a range (ones complement)
F9	Calculate/Change checksum (8-bit)
F10, Esc, Enter	Exit the editor

Table 1 – Memory Data Editor Keyboard Usage

SELECT ADDRESS RADIX (F2)

This key toggles between hex and decimal addresses on the left side of the screen.

SET CURSOR ADDRESS (GOTO) (F3)

Type the address to view or edit, choose HEX or DECIMAL before typing the address.

RECONFIGURE (F4)

Reconfigures the editor display. Type one letter for each radix you want displayed on-screen. Any combination of displays is acceptable. The four radixes available, ASCII, binary, hex, and octal, are represented by the letters **a, b, h,** and **o** (*e.g.*, for binary only type <**b**>, and <**Enter**>. For hex, ASCII, and octal, type <**hao**>, and <**Enter**>.)

SEARCH FOR PATTERN (F5)

Searches the data buffer for a specific pattern. The pattern can be entered in ASCII or hex. To enter an ASCII string, type an apostrophe (‘), then the string with no extra spaces. To enter a hex pattern, just type the hex pattern you want to find. Then, specify the address where the search starts and whether to search toward higher (**FORWARD**) addresses or toward 0 (**BACKWARD**).

FILL RANGE (F6)

Specify the byte-range to fill (inclusive) and the data to be placed in those bytes.

COPY (F7)

Specify the data source byte-range and the destination address for the first byte. The destination address may overlap the source address. This operation is similar to an insert or deletion operation.

***Example**: Copying from 0-FF to address 1 will "insert" a new location at 0, shifting the existing data down one address.*

INVERT A RANGE (ONES COMPLEMENT) (F8)

Perform a ones complement on a range of addresses. A ones complement uses the formula: newbyte = 255 - oldbyte. Complementing a byte twice restores its original value.

CALCULATE/CHANGE CHECKSUM (F9)

Add all, every other or every fourth byte over the specified range of addresses and then set a particular address to generate the desired checksum. If you have the "*Checksum method:*" under the *Buffer/Options* command set to 16-BIT you will also be allowed to specify the byte order, as seen in the example below. After setting the parameters below and hitting <**Enter**>, the current bit sum of the 8 (or 16) bit wide values is displayed in a dialog box. The box lets you change a byte (or a word for 16-BIT sums) to modify the sum.

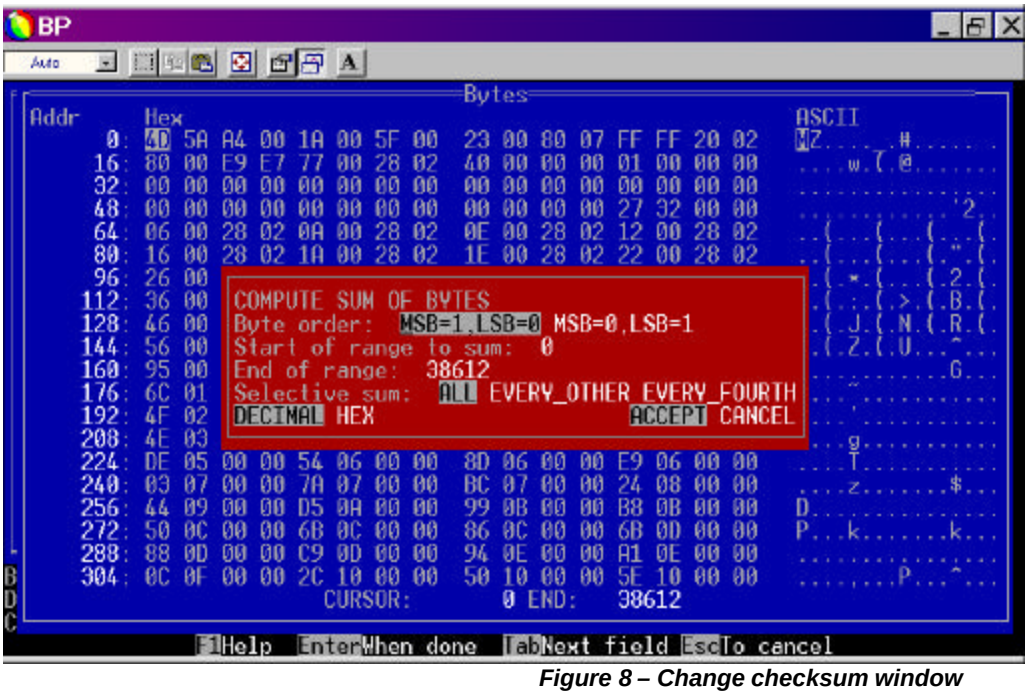


Figure 8 – Change checksum window

Changing any byte in the range changes the checksum; the last two (or four) digits of the sum can be set by changing a byte (or a word). To change the sum, decide which

address to change and specify a new sum. Users will often set an EPROM sum to zero so their microprocessor can determine if the EPROM data is valid. The *"Selective sum:"* option is provided so you may set the sum for the even bytes and then set the sum for the odd bytes. This is useful when multiple chips are used in the target circuit and each chip must have its own checksum. To do this you must execute the command once with a particular start address and then execute it again with one plus the same start address.

EXIT THE EDITOR (F10, ESC, ENTER)

All changes to the data while in the editor are permanent regardless of how you exit the editor. You may think <**Esc**> will not save the changes, but it does. The best way to get back the original data is to load the buffer again by reading a chip or loading a file.

FUSE DATA EDITOR

When the *Buffer/Edit* command is used on a PLD, the display shows JEDEC fuse data where a 0 indicates an intact fuse (a connection in the PLD array) and a 1 indicates a blown fuse (no connection). The number to the left of each row shows the starting address for that row, with fuse locations numbered sequentially from left to right. Fuse numbers start at 0, so fuses in a device with 2048 fuses are numbered 0 to 2047. The total fuse number in the buffer is END address + 1.

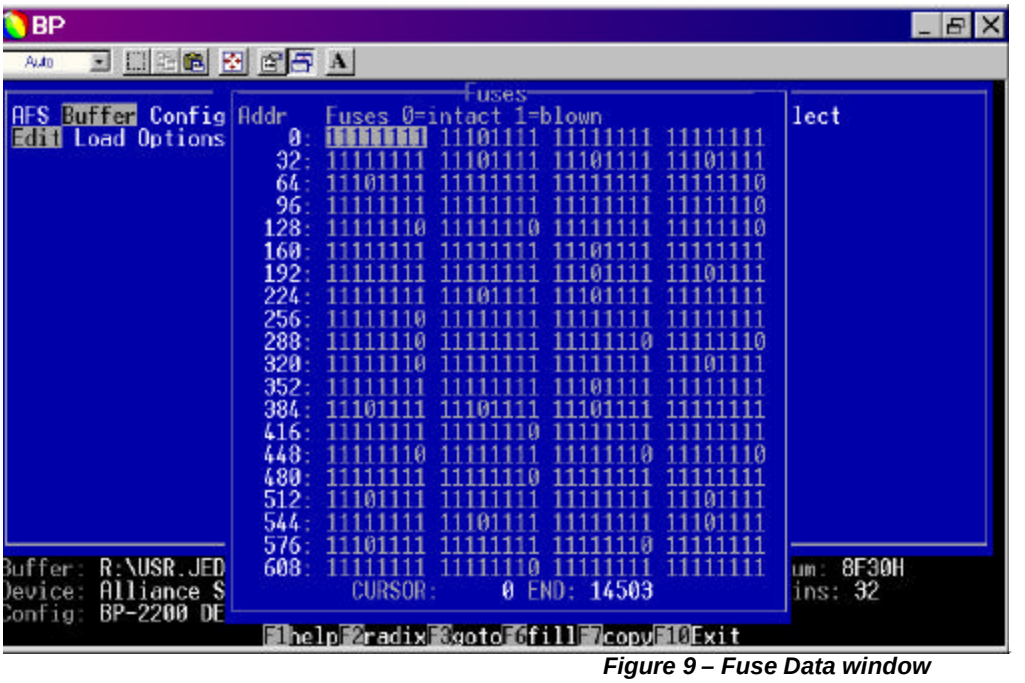


Figure 9 – Fuse Data window

Keyboard Usage:

¬, - ®, ¯ PgUp, PgDn	Move cursor
Ctrl ¬, Ctrl ®	Move one digit left or right
Ctrl-PgUp	Go to top of buffer

Ctrl -PgDn	Go to end of buffer
F1	Help on key usage
F2	Select address radix (decimal/hex)
F3	Set Cursor Address (goto)
F7	Copy a range of data
F10, Esc, F10	Exit the editor

Table 2 – Fuse Data Editor Keyboard Usage

The function keys for this editor operate as described above for the Memory Data Editor.

TEST VECTOR EDITOR

Each row of the display shows one test vector, with one character for each pin on the chip; pin 1 at the left, the last pin at the right. Test vectors are numbered starting with 1, so **END** indicates the total number of vectors stored in the buffer. The **CURSOR** location shows both the vector number (followed by a colon) and the pin number where the cursor is located. Below is an example of 24 pin vectors with the cursor on pin 1 of vector 1:

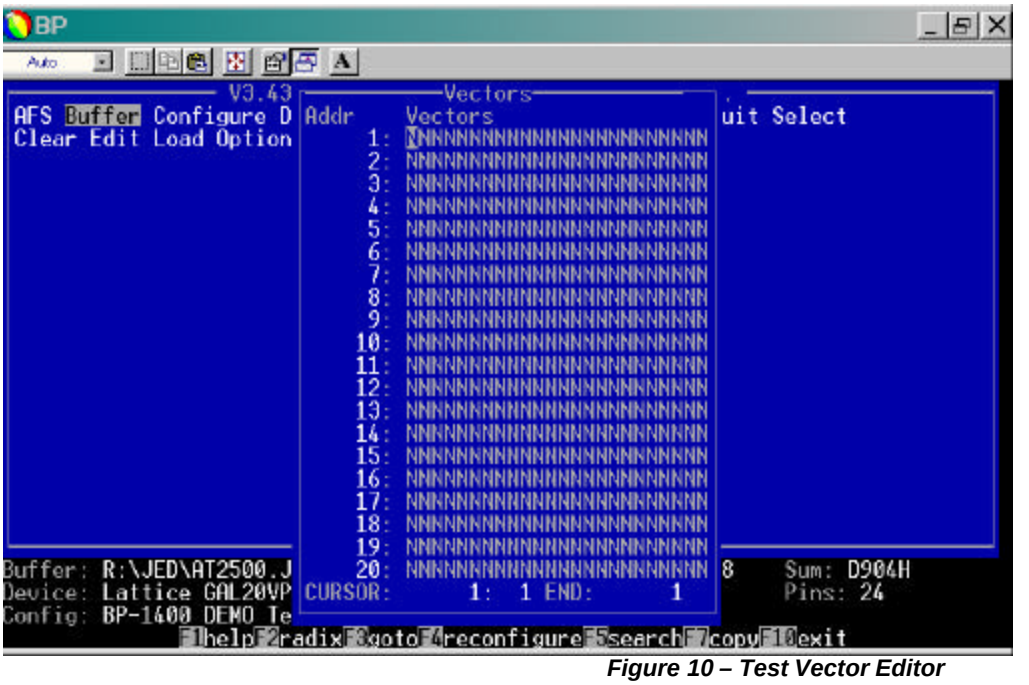


Figure 10 – Test Vector Editor

Keyboard Usage:

¬, - ®, ¯ PgUp, PgDn	Move cursor
Ctrl PgUp	Go to top of buffer
Ctrl PgDn	Go to end of buffer
F1	Help on key usage

F2	Select address radix (decimal/hex)
F3	Set cursor address (goto)
F7	Copy a range of data
Esc, F10, Enter	Exit the editor

Table 3 – Test Vector Editor Keyboard Usage

The function keys for this editor operate as described above for the Memory Data Editor.

CHAPTER 5
FILE FORMATS

FILE LOAD FORMAT EQUIVALENTS

The BP.EXE software loads many different file formats and automatically recognizes these formats. Nine formats appear in the *Buffer/Load* dialog box, but many of these are general purpose translators that can read multiple formats. File types are automatically determined when the software scans the file, so you don't have to become an expert on file formats.

BP Microsystems supports new formats when requested by customers. Many of the formats in the following table are obsolete and have never been requested. If you need support for a format that is not in this table, please contact our Technical Support staff so we can satisfy your needs.

This list includes the Data I/O format numbers along with the corresponding BP format to choose.

File Format	Data I/O Number	BP Format Designation and Status
ASCII-BNPF	01, 05	obsolete [1]
ASCII-BHLF	02, 06	obsolete [1]
ASCII-B1OF	03, 07	obsolete [1]
Texas Instruments SDSMAC 320	04	SDSMAC(320)
5-level BNPF	08, 09	obsolete [2]
Formatted Binary	10	FormatBin [2]
DEC Binary	11	obsolete [2]
Spectrum	12, 13	not supported [3]
POF	14	POF
Absolute Binary	16	BINARY
ASCII-Octal Space	30, 35	obsolete [4]
ASCII-Octal Percent	31, 36	obsolete [4]
ASCII-Octal Apostrophe	32	obsolete [4]
ASCII-Octal SMS	37	obsolete [4]
ASCII-Hex Space	50, 55	ASCIIHEX
ASCII-Hex Percent	51, 56	ASCIIHEX
ASCII-Hex Apostrophe	52	ASCIIHEX
ASCII-Hex SMS	57	ASCIIHEX
ASCII-Hex Comma	53, 58	ASCIIHEX
RCA Cosmac	70	obsolete
Fairchild Fairbug	80	FAIRBUG
MOS Technology	81	obsolete
Motorola Exorciser	82	MOTOROLA
Intel Intellec 8/MDS	83	INTEL
Signetics Absolute	85	obsolete
Tektronix Hexadecimal Format	86	TEKHEX
Motorola EXORmacs	87	MOTOROLA
Intel MCS-86 Hexadecimal Object File Format	88	INTEL
Hewlett-Packard 64000 Absolute	89	not supported
Texas Instruments SDSMAC	90	SDSMAC
JEDEC format (Full)	91	JEDEC
JEDEC format (Kernel)	92	not supported
Tektronix Hexadecimal Extended	94	TEKHEX
Motorola 32 bit (S3)	95	MOTOROLA
Hewlett-Packard UNIX format	96	not supported
Intel OMF 386	97	OMF
Intel OMF 286	98	OMF
Intel Hex-32	99	INTEL

Table 4 – File Formats

[1] This file format was designed for easy human input before binary editors were available.
[2] This format was designed for use with paper-tape readers and is now obsolete.
[3] This is a very verbose ASCII binary format that has not yet been requested by customers.
[4] This is an unpopular format that has not yet been requested by customers.

HEX FILES

Hex files and binary files are used to store data that can be programmed into a chip. The hex files are ASCII files containing hex characters (0-9, A-F) and other information. Information on editing hex files can be found in *Chapter 4, Using the Data Editors*.

Straight-hex Files

The straight-hex file format is the simplest. It consists of two hex characters for each data byte in the file. The hex characters are separated into lines with a Carriage Return, Line Feed sequence. The file contains neither address information nor any checksums.

Hex-space Files

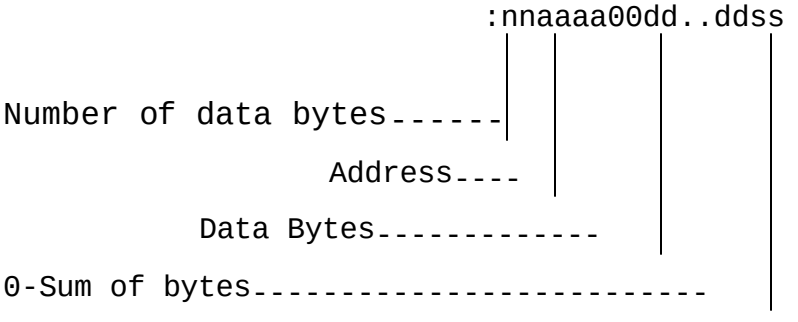
The hex-space format is identical to straight-hex format except spaces may be placed between hex pairs representing bytes.

Intel Hex Files

BP programmers support both MCS80 and MCS86 style Intel hex files. The MCS80 file format uses 16-bit addresses and is therefore limited to 64k bytes (21^6 bytes). The newer MCS86 format (extended Intel format) adds an address offset record that extends the file addresses up to 20 bits, 1 Megabyte.

MCS80

Each MCS80 data record is formatted as follows:

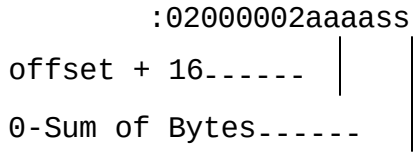


The end record is:

```
:00000001FF
```

MCS86

MCS86 files use an address offset record. The record specifies an offset to add to subsequent data records:

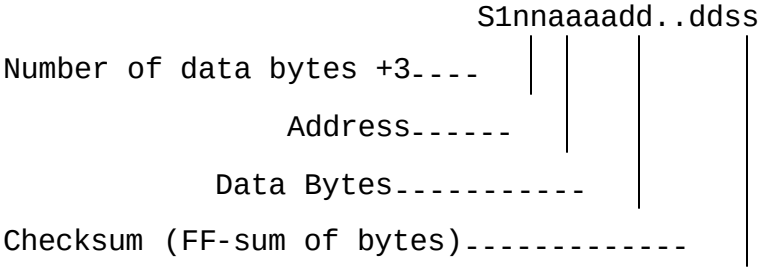


Motorola Hex Files

Motorola hex files support 16-, 24- and 32-bit addresses. The size limitation of this file format is 4 gigabytes. The BP-4100 supports the full range of record numbers, S0-S9.

S1

S1 data records (64KB limit) are:

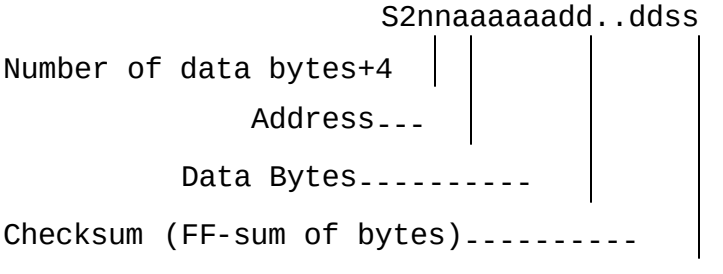


The end record is:

```
S9
```

S2

S2 data records (16MB limit) are:

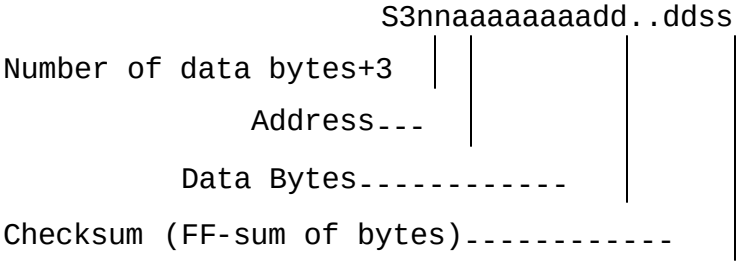


The end record is:

```
S8
```

S3

S3 data records (4GB limit) are:

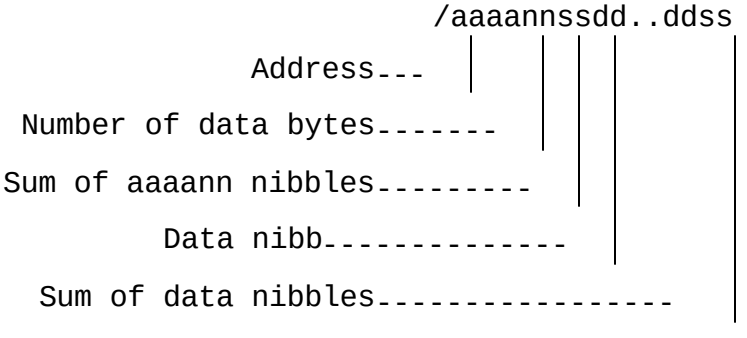


The end record is:

```
S7
```

Tekhex Files

Tektronix has their own hex format, Tekhex. It supports only 16-bit address records.



The end record is:

```
/00000000
```

Addresses

The Intel, Motorola and Tekhex files contain an address in each line of data. This address tells the BP-4100 which bytes to program with the data on the line. Straight hex files and binary files do not contain addresses. BP Microsystems programmers count each byte to generate an address, i.e., the first byte is 0, the next is 1 and so forth. Thus if you move or append lines to a straight hex file, the resultant EPROM will be different. If you move the lines around in an MCS80 Intel, Motorola or Tekhex file the data will not be changed.

Checksums

Intel, Motorola and Tektronix hex files include a checksum in every line. This code is calculated from the data on the line. When the programmer receives a line, it recalculates the checksum and compares it to the one from the file. If there is an error in the file (such as missing characters) the sum will not be correct and you will get the message:

```
        Warning--Checksum Error:

    line causing error is printed here
```

In this case, the programmer assumes the address and data information is correct anyway and program normally. If you edit the data fields in a hex file without updating the checksum, you will get this message.

CHAPTER 6
BP SOFTWARE COMMAND REFERENCE

This chapter describes the commands provided by the BP DOS interface software. Commands are located in menus at the top of the screen. To execute a command, type the first letter of the menu entry or use the cursor keys on the keyboard to select the desired command and press <**Enter**>. In this chapter, command names are written as *Name* or *Menu/Name* where *Menu* is the menu name, if used, and *Name* is the entry that invokes the command.

Some commands may not appear on your particular programmer model. For example, if your programmer only supports PLDs, the commands used strictly for memory devices will not be present in your interface software and the converse is true if your programmer only supports memory devices. Furthermore, sometimes identical commands invoke different routines when programming memory devices versus PLDs. For an example, see the *Device/Secure* command. Some menu screens and commands appear only for specific chips. The *Device/Configure*, *Device/UES*, and *Device/Encrypt* commands are examples of commands that only appear on applicable devices.

KEYBOARD USAGE

The following keys may be used at any time:

F1	Help
Esc	Interrupt a command or return to the previous menu

The program starts at the command level. These keys operate any time you are at the command level.

®, ¬, Space	Move cursor, highlight selection
¯, Enter	Execute the command that is selected
A-Z	Select and execute command
-, Esc	Move up one line in the command menu
F2	Chip Information
F3	Hot-Key Help

Tables 5 & 6 – Common Keyboard Usage

"Hot-Keys" also operate at the command level:

Alt-C	Configure
Alt-D	Diagnostic h/w test
Alt-E	Buffer/Edit
Alt-H	Help Menu (F3)
Alt-I	Chip Information (F2)
Alt-L	Buffer/Load
Alt-O	Device/Options
Alt-P	Device Program
Alt-Q	Quit
Alt-R	Device/Read
Alt-S	Select
Alt-T	Device/Test
Alt-V	Buffer/Vectors
Alt-X	Quit
Alt-0…9 **Alt-F1…F9**	Play macro file #.pgm (e.g., Alt-1 plays macro file 1.pgm)

Dialog boxes gather information before executing commands. The dialog box keys are:

Esc	Cancel dialog box and return to command level
Enter	Exit dialog box and execute command
Tab	Move to next field in dialog box
Shift-Tab	Move to previous field in dialog box
Home	Move to first selection on a line
End	Move to last selection

When editing text:

Backspace	Delete character to left
®, ¬	Move cursor
Home	Move to left end of line
End	Move to right end of line
Ctrl-U	Clear line
Ins	Turn ON/OFF insert mode

When selecting a file (under Buffer/Load or Buffer/Save) or chip (under Select) from a list, editing the selector string (directory or part number) changes the list. Highlight the item you want and press <**Enter**>. To make selections:

¯, -, PgUp, PgDn	Move selector highlight
Enter	Make selection

Tables 7, 8, 9, & 10 – Common Keyboard Usage

AFS

Advanced Feature Software is additional software that can be purchased to run with the BP Software used for the programmer. This software includes Serialization, JobMaster, Advanced Support, and Handler software (needed anytime an autohandler is used).

AFS/SERIALIZE

Description

Serialize devices programmed.

Application

Use this command to select a serialization pattern. Choose from **NONE**, **Simple** and **Complex**.

Operation

Select *AFS/Serialize* and choose an option.

See Also

Chapter 10, Serialization, page 10-1

AFS/UPGRADE

Description

Enter an Upgrade code for additional software purchased for use with the BP Software.

Application

Once you have contacted BP Microsystems, Inc. and purchased additional software, i.e. JobMaster, an upgrade code will be generated and sent to you. This code is used to gain access into the additional software provided with your software version.

Operation

Select *AFS/Upgrade* from the BP Software main menu. At the window prompt, type in the Upgrade code provided by BP Microsystems, Inc. and select ***Accept***. Once accepted, the BP Software will begin upgrading the programmer sites. When the upgrade is complete, you will be prompted you to re-boot your system to initialize the AFS you have purchased.

BUFFER COMMANDS

BUFFER/CLEAR

Description

Clears all memory, fuse, and test vector data buffers.

☐ *This command is only available when the "User Mode" is set to EXPERIENCED under the* Configure *command.*

Application

Use this command to empty the data buffers before using one of the edit commands to create new data (*Buffer/Edit*, or *Buffer/Vectors*).

Operation

Press **<Enter>** to clear the buffers. Press **<Esc>** to cancel the command.

☐ *The buffers are automatically cleared during a* Buffer/Load *command if the "Clear buffer before loading" option is set to YES under the <**F8**> additional options in the* Buffer/Load *dialog box.*

See Also

Device/Read, Buffer/Load, Device/Options

BUFFER/EDIT

Description

Edit the buffer contents.

Application

Use the data editor to examine or modify patterns read from a file or a chip. This data may then be programmed into a chip (*Device/Program*) or stored in a file (*Buffer/Save*).

Operation for memory devices

The editor screen will appear in hex and ASCII formats when a memory device has been read or a corresponding file has been loaded. It can be reconfigured to show octal and binary formats. A flashing cursor appears on the active side under the character that can be edited, and is highlighted on both sides of the display. To change the character, press any valid character for the active data format (0-9, A-F for hex, any printing characters

for ASCII, 0 or 1 for binary, and 0-7 for octal). Use the cursor motion keys to scroll through the buffer.

- *Remember that <**F1**> will give you help. The other function keys (**F2-F10**) allow you to reconfigure the editor, edit data, or exit the editor.*

Operation for PLDs

When a PLD has been read or a JEDEC file has been loaded, the editor comes up in a binary format representing the fuses in the device. Use the cursor keys to scroll through the buffer. Type 1 or 0 to change a fuse. Press <**Enter**> or <**Esc**> when finished.

- *Editor data corresponds to 'L' fields data in the JEDEC file.*

- *The term "fuse" is accurately used when referring to fuse-link programmable parts. During programming, these parts actually melt a metal link that carries a signal when the chip is in operation. EPROM and EEPROM based PLDs, however, don't use fuses. They store a programmed bit by placing electrical charges on a MOS transistor gate. "Fuse", in this case, is a historical carry-over from the older technology.*

See Also

Chapter 4, Using the Data Editors, page 4-1

BUFFER/LOAD

Description

Load a data file from disk into the buffer.

Application

Load the buffer before programming. Once loaded, the file data may be edited and saved to disk.

Operation for Memory Devices using Hex/Binary Files

A hex or binary file describes the data to be programmed into a memory chip. Hex file standards define consistent ways to represent data, contain checksums to verify file integrity, load addresses that specify where to place data in a memory chip or system and can be printed. They are often generated by compilers and assemblers to program EPROMs. Any of the file formats can represent data to be programmed into any chip. The *Buffer/Load* command will translate the file into a binary representation stored in the buffer.

A selector box will appear showing you the contents of the directory specified in the *Directory* field at the top of the dialog box. You can edit this line to see files in a specific directory, such as C:\files*.hex.

Selecting File Name and Type

Select the desired drive, directory and file extension at the *Directory* prompt. Use *.* to show all files.

Use the arrow keys to move the cursor down until it is over the desired file name, and then press <**Enter**> to select it. The specified file name will appear to the right of the *File to load* field and the file type is automatically determined in the *Type* field.

- *The software almost always determines the correct file type, but it looks at only the first 1K of the file, so JEDEC files with large comment blocks at the beginning and other similar situations may fool the software. If the software chooses the wrong type of file and you wish to choose another particular file type, you can override the "Type" by using the <**Tab**> key to get to that field and then selecting the desired type with the arrow keys.*

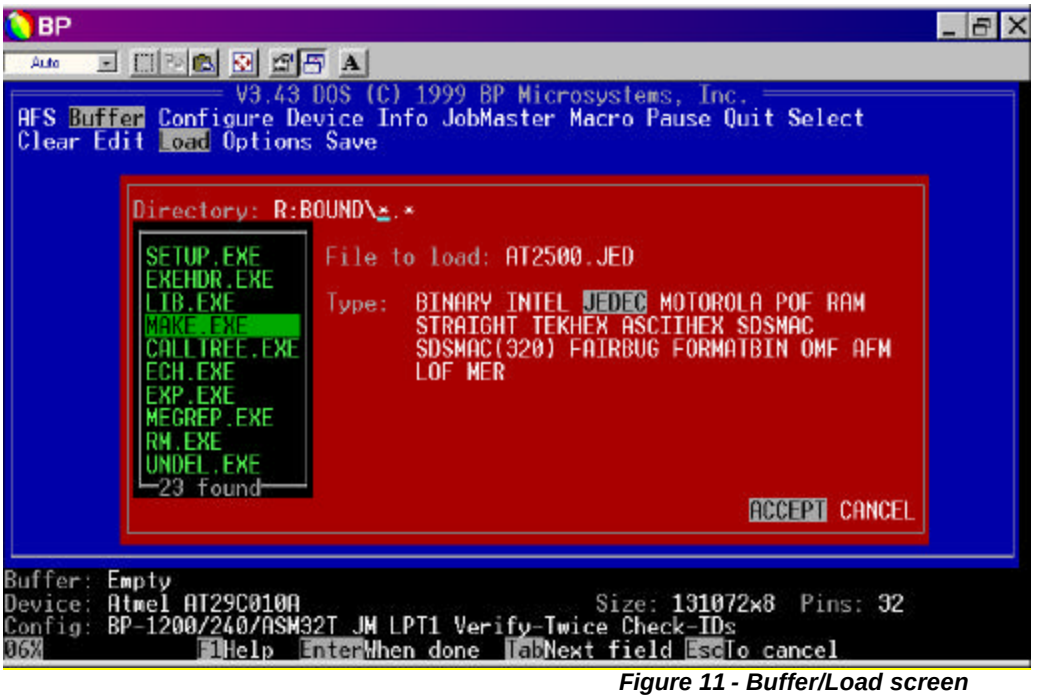


Figure 11 - Buffer/Load screen

If you wish to turn off *Automatic file identification*, then simply go to the *Buffer/Options* command and disable that option.

Alternatively, if *File to load* is blank, you can type the exact path and file name on the *Directory* line. After typing the file name, press <**Enter**>.

If you press <**Enter**> once again, the default parameters load the entire file starting at buffer address 0; however, you may wish to control how the data is loaded into the buffer. If so, then use the <**F8**> for additional options (listed below).

If you do a *Buffer/Load* again and want the same file, then just press <**Enter**> twice.

F8 – Additional Options

"Load parameters" control where the file is loaded in the buffer and how much of the file will be loaded.

Press **<F8>** to see the additional load parameters.

Normally, you will want to clear the buffer, but to combine multiple files you may specify loading a new file without clearing the buffer. Use the arrow keys to change the *Clear buffer before loading* field.

A common requirement is loading a file generated by an assembler or compiler with a starting address that is other than zero.

> ***Example:*** *If your code executes at address F0000 hex, you will specify that address as the* Lowest address to load *parameter. For hex files, the software automatically places the first address offset encountered in the file as the* Lowest address to load.

For most purposes, this is what you want, but some compilers generate the code in random order, which will make this feature less desirable. To turn off "*Automatic lowest address to load:*", go to the *Buffer/Options* command and disable this feature.

The *Highest address to load* field may be used to discard any data from your file with a higher address, rather than loading it into the buffer.

Another common requirement is to load multiple files into the buffer. In this case, specify the *Load address in buffer* parameter where the first byte is to be loaded into the buffer. Specifying both lowest and highest addresses to load will exclude data from the file outside that range. Remember, with the first file you load you will want the *Clear buffer before loading* option set to **YES**, but on the subsequent files you will want to set it to **NO**.

Press the <**Enter**>key to continue the load process.

The file will load into the buffer. If any errors are detected, messages appear describing the problem. Since hex files may have a checksum on each line, a checksum error can result if the checksum doesn't match the sum of the bytes on the line. This is usually the result of editing the file without updating the sum. If this happens, you will receive a warning message. The file will still load – the message is there alerting you to a potential problem.

A message will appear on-screen showing the number of bytes loaded from the file, how many bytes the file contains, and how many lines the file contains (hex files only). Also, the buffer status line at the bottom of the screen will display the number of bytes in the buffer, the checksum and the file name.

Operation for PLDs using JEDEC files

JEDEC files are the standard method of describing PLD use patterns and test vectors. The file format standard was set up by the Joint Electron Device Engineering Council (JEDEC). JEDEC files may contain fuse data, test vectors, part numbers, checksums, and a secure enable option. The file's checksum lets the software verify that a given file is intact and has not been unintentionally modified. JEDEC files normally use the extension (last three letters) ".JED", but frequently ".JDC" or ".J01" is also found.

The data in any JEDEC file can be used to program and test any PLD, but useful results are obtained only when the data is used to program a chip with the correct organization or architecture. Generally, all devices with the same generic name (*e.g.*, 16L8) may be programmed from the same JEDEC file to produce functionally identical parts. Three exceptions are the 16V8, 20V8, and 20R8 devices. The Signetics 16V8 and 20V8 differ from the GAL16V8 and GAL20V8. There are two varieties of 20R8s – one has ten outputs and the other has eight.

1. A selector box appears showing the contents of the specified *Directory*. You can change the directory specification to see files in a different directory. Normally, the directory specification should end in ".JED", such as "C:\FILES*.JED".

2. Select the file name.

3. Select the desired drive, directory and file extension on the *Directory* line.

4. Move the cursor until it is over the desired file, then press <**Enter**>. The desired file name is placed in the "*File to load:"* field and the JEDEC file type should have automatically been selected. If it was not selected and you are sure the file you specified is a JEDEC file then use the arrow keys to force it to load as a JEDEC file.

5. Press <**Enter**> again to load the file.

6. Alternatively, if the *File to load* line is blank, you can type the exact path and file name on the *Directory* line. After typing the file name, press <**Enter**> twice.

7. If you want to load the same file again, just press <**Enter**> twice after executing the *Buffer/Load* command.

The software clears the buffer, shows any starting comments in the file, and loads the file into the fuse and vector buffers. The automatic secure option will be set as specified by the file (see *Device/Options*). If a part number is specified in the file, it will automatically appear as the device the next time you use the *Device/Select* command.

The buffer status line at the bottom of the screen shows how many fuses and vectors were loaded, the fuse checksum, and the file name.

Loading files from floppy disks

When using a hard disk, displaying the current directory on-screen takes a small amount of time. However, the display time may become rather slow when using a floppy disk. To speed up the display, use the *Buffer/Options* command and set the *Show file names* option to **DISABLE**. The *Buffer/Load* dialog will then appear similar to the following:

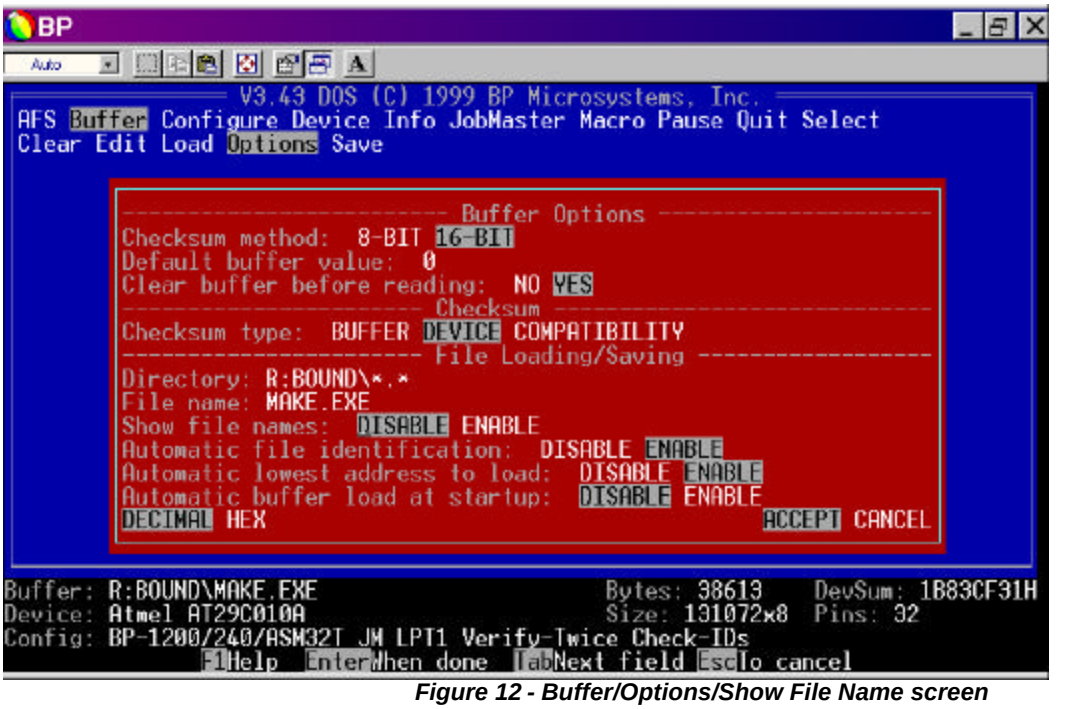


Figure 12 - Buffer/Options/Show File Name screen

When the file name display is turned off, you have to type the exact file name and press <**Enter**> to load the file.

See Also

Buffer/Options, Buffer/Save

BUFFER/OPTIONS

Description

Set the options that control the editing, loading, and saving of the buffers.

Application

Select generic buffer options such as: (1) checksum method; (2) default value; and (3) whether or not to clear before each *Device/Read* command. Also, set several options pertaining to the *Buffer/Load* and *Buffer/Save* commands, which include: (1) changing the default directory; (2) changing the default filename; (3) disabling the listing of files in a selector box; (4) disabling automatic file type identification; (5) disabling the automatic setting of the lowest address to load; and (6) enabling the loading of the filename when the software starts up.

Operation

A dialog box will appear showing the current settings for the buffer options.

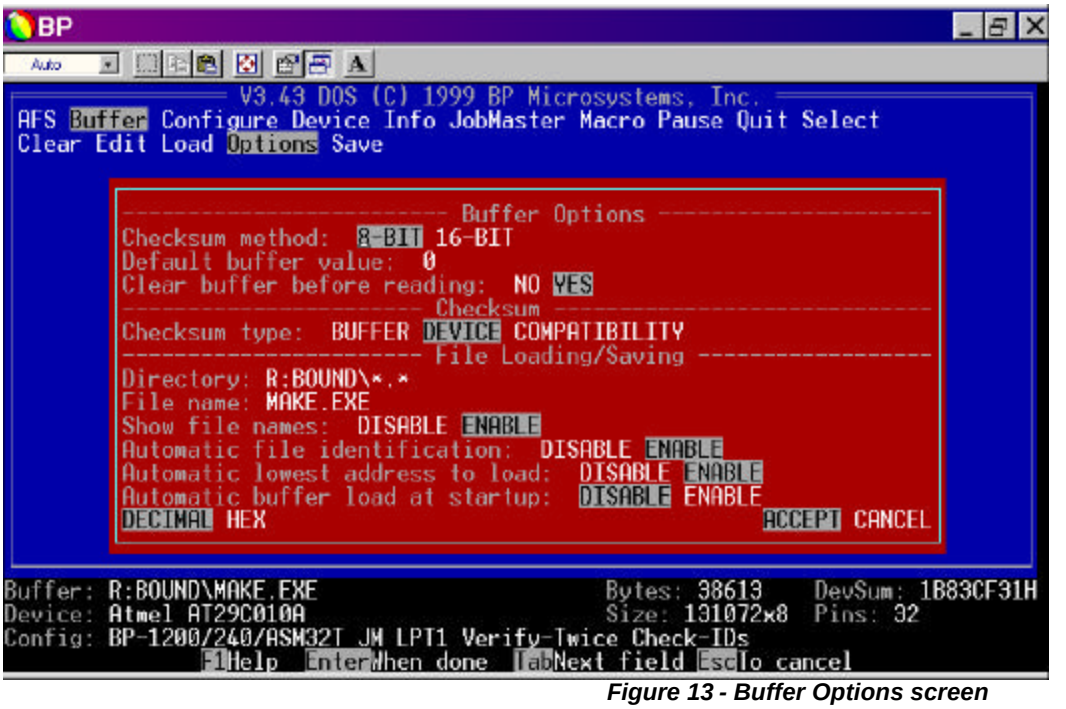


Figure 13 - Buffer Options screen

Use the <**Tab**> key to select the field you want to change.

Press the <**F1**> key on any field to get context-sensitive help.

Change selections using the <**Left**> and <**Right**> direction keys.

Press <**Enter**> when finished.

Checksum method

This dictates how the checksum is computed. Adding 8 bits at a time is the default, but you can specify 16 bits at a time instead. This affects the checksum displayed at the

bottom of the screen and also affects the *checksum* command, <**F9**>, when editing hex or binary files. *See Chapter 4, Using the Data Editors, page 4-1.*

Example: *A 27256 EPROM contains 32,768 eight-bit bytes, or 32K-bytes. A 27210 contains 65,536 sixteen-bit words. Most EPROMs come in two widths: byte-wide (8-bits) and word-wide (16-bit). Multiplying the data width by the number of words gives the size of the array, in bits. The 27256 contains 262,144 bits (256K-bits); the 27210 contains 1,048,576 bits (1-megabit).*

Default buffer value

This is the value that the buffer uses when it is cleared. The buffer is cleared when a new file is loaded, if the *Clear buffer before loading* option is set to YES in the *Buffer/Load* dialog box; or when a chip is read, if the *Clear buffer before reading* option is set to YES in the *Device/Options* or the *Buffer/Options* dialog box. When the software is first installed, the default value is set to zero (0), which makes calculating the checksum very fast because the starting checksum is assumed 0 and added from that point when data is changed in the buffer. This allows calculation of the checksum without having to know the range of data in the buffer.

If you set this default value to anything other than 0, then you will no longer see the checksum displayed at the bottom right of the screen. To calculate the checksum you will need to use the checksum **<F9>** command under the *Buffer/Edit* command and specify the desired range.

Clear buffer before reading

The *Clear buffer before reading* field is defaulted to YES, since you will typically wish to see only the contents of the chip in the programmer site. However, if you wish to read portions of different chips and combine them into one file, then you will want to set this to NO and possibly use the *range of fields* in the *Device/Options* command.

Directory

This is the currently selected directory last used by the *Buffer/Load* and *Buffer/Save* commands.

File name

This is the file name used by the *Buffer/Load* and *Buffer/Save* commands and will be saved when the configuration is saved according to the *Configure* command.

Show file names

This dictates whether or not a list of files and directories is shown in a selector box when you use the *Buffer/Load* and *Buffer/Save* commands. The purpose of this option is to

speed up file retrieval on PCs with slow disk access when you already know the exact file name you wish to use.

Automatic file identification

The software is shipped with this option **ENABLED** to help the user select the proper file type when loading a file with the *Buffer/Load* command. This does require disk access and may slow processes down minutely, but still assists in recognizing most file types and therefore allows you to continue forward once a file is selected without having to figure out what type of file has been chosen and inputting that information. It may also be annoying if it selects a file type other than what you really want.

Example: *You may have a file that should really be loaded as a BINARY file, but it has some characters at the beginning of the file that make it look like one of the other file types (this is very unlikely, but it could happen). In that case, you will probably prefer to DISABLE this option to avoid having to change the file type every time you execute the* Buffer/Load *command.*

Automatic lowest address to load

This feature tries to guess what the *Lowest address to load* is for the *Buffer/Load* command when using a hex file. When **ENABLED** it simply reads the first address offset specified by the hex file and assumes that it is the desired offset.

This is usually a good assumption for files produced by the programmer or most assemblers and compilers; however, some files do not begin with the first byte to be placed into the buffer; they may actually place the data in a non-sequential order in the hex file or intend to leave the beginning of the buffer empty. If so, this feature should be turned off, because it will probably insert the wrong lowest address to load and consequently not load all the data into the buffer or place the data at the wrong address.

Automatic buffer load at startup

When **ENABLED**, this option will load the file specified by the *Directory* and *File name* above, when the software is first started. This is a simple alternative to creating a macro with the *Macro/Record* command, which does the same thing.

See Also

Buffer/Edit, Buffer/Load, Buffer/Save, Device/Options, Macro/Record

BUFFER/SAVE

Description

Save data buffer contents to disk.

Application

Use this command to make a new hex, binary or JEDEC file, or to save a file after editing.

Operation

A selector box appears showing files in the specified directory (similar to the *Buffer/Load* command). To overwrite one of these files, select it with the cursor keys and press <**Enter**> twice. To create a new file:

- Specify the desired directory after the *Directory* prompt.
- Press <**Tab**> to move to the *File to create* field. Type the name of the file you wish to create.

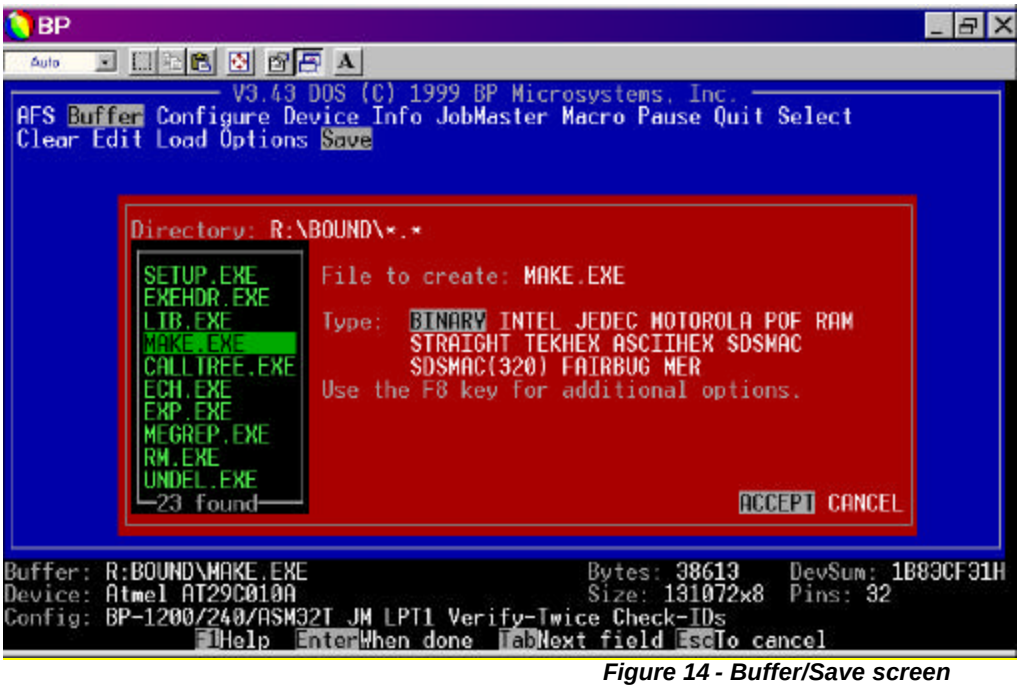


Figure 14 - Buffer/Save screen

- *Press <**Tab**> and the arrow keys to specify a particular file type. For the non-JEDEC and non-POF files you can press <**F8**> to set the* Lowest address to save *and the* Highest address to save. *Of course, the default is to save the entire file to disk.*

- *Press <**Enter**> to save the file.*

- *If a JEDEC file is created, it will contain the current contents of the fuse and vector buffers and the name of the selected chip, if any. The current X-value and any automatic secure options are also indicated in the file. A comment is inserted at the beginning of the file indicating the present time, date and where the data is coming from.*

☐ *Attempting to overwrite an old hex, binary or JEDEC file will cause the old file to be renamed* filename.*BAK*.

See Also

Buffer/Options, Buffer/Load, Device/Options

BUFFER/VECTORS

Description

Edit or view the test vectors for PLDs.

☐ *This command only appears when a PLD is selected.*

Hot-Key **Alt-V**

Application

Examine or modify test vectors to debug new designs. This is the vector portion of the JEDEC or POF file contents.

Operation for PLDs

Use the cursor keys to scroll through the buffer. Test vectors may be changed by typing over the original data in the buffer. Valid characters for the test vectors are **0, 1, B, D, C, F, H, K, L, N, P, U, X, Z,** and **?**. Press <**Esc**> when finished.

- The data in the editor came from the line in the JEDEC file that starts with the letter **V**.

- To create test vectors from scratch, specify the chip first so the vector editor knows the correct number of chip pins.

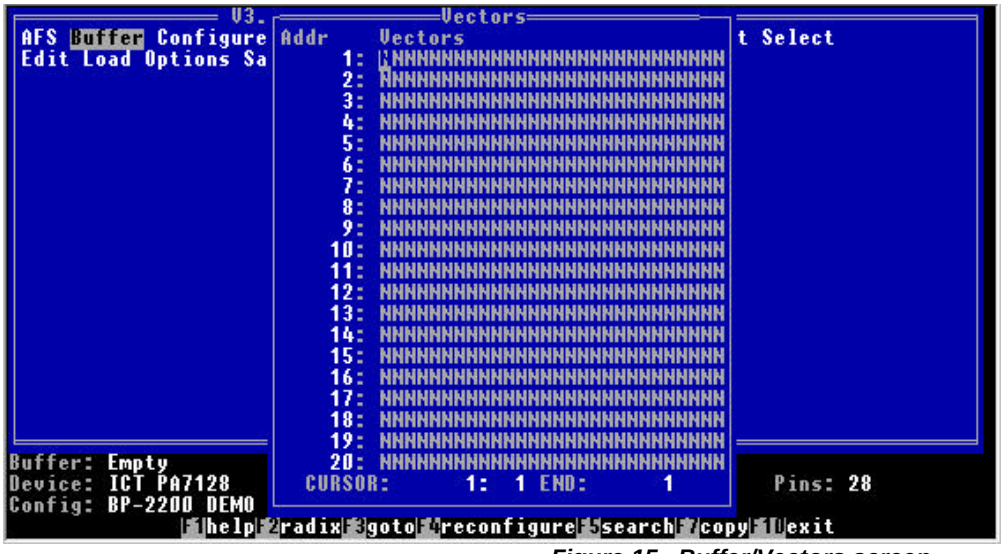


Figure 15 - Buffer/Vectors screen

See Also

Chapter 4, Using the Data Editors; *Test Vectors, Buffer/Load*

CONFIGURE COMMANDS

Description

Select the configuration options that control operation of the programmer.

Hot-Key **Alt-C**

Application

Select other operating modes, such as:

- which programmer you wish to communicate with
- whether you have a color or monochrome display
- which parallel port you want the software to look at first when it starts up
- when to save the configuration options that have been set under this *Configure* command and under the *Buffer/Options, Device/Options,* and *Device/Handler* commands
- experienced or novice mode of operation; and (6) whether the startup messages appear

Operation

A dialog box will appear showing the current settings for the startup configuration options.

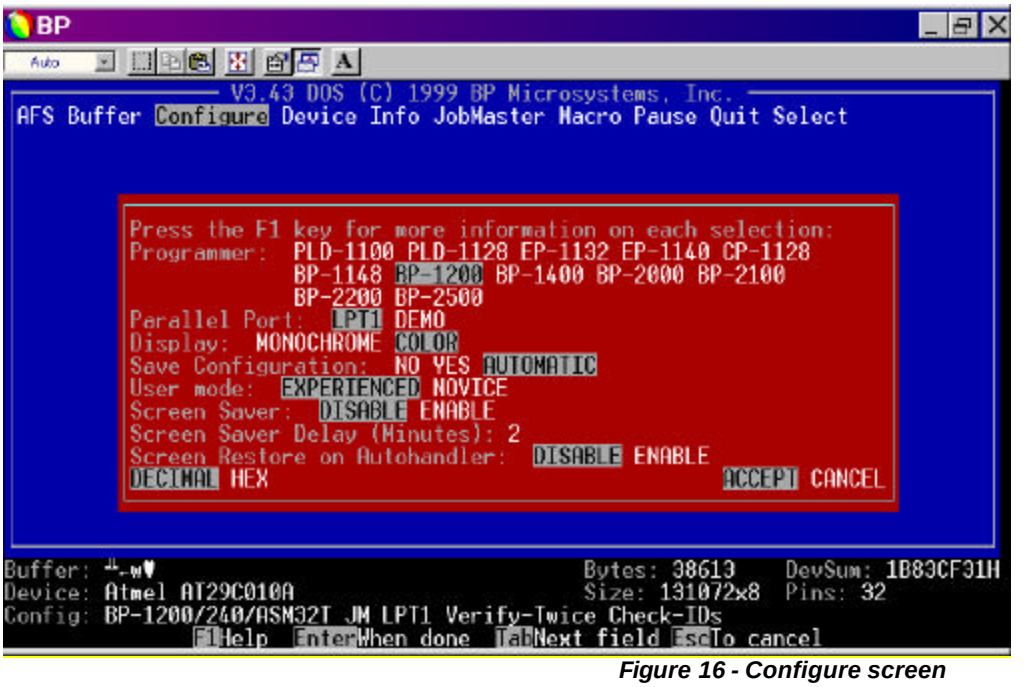


Figure 16 - Configure screen

- Use the <**Tab**> key to select the field you want to change.
- Press the <**F1**> key on any field to get context sensitive help.
- Press <**Enter**> when finished.

To make a permanent change, you must select YES or AUTOMATIC in the Save/Configuration *field.*

Programmer

When the software is started, the correct programmer type and parallel port are identified in the *Programmer* and *Parallel Port* fields; however, you may change the programmer type if you wish to see the different device support for the other programmers. It will simply put you into a demonstration mode and pop up a warning any time you execute a command that requires the presence of a programmer.

If a programmer has not been detected, the software will automatically start up in DEMO mode, allowing the user all functionality possible without a device programmer. A programmer can be added after startup in DEMO mode through the *Configure* option. All actions/functions found in the software can be utilized within DEMO except programming a device.

Parallel Port

This specifies which port to attempt communication with a BP Microsystems programmer when the software starts up. If a programmer is not found on the specified port, then the software scans other available ports and tries to find a programmer somewhere. If no programmer is found, then it puts itself into a demonstration mode that allows access to all the software features that do not require the presence of a programmer.

Display

When using a color video adapter (CGA, EGA, or VGA), you can choose a black and white display, if desired. If you have a monochrome monitor it is not necessary to change this field.

If you are using a laptop, it may be easier to read the screen if you choose a black and white display.

Clear Screen after Commands

ENABLE will clear the screen after each command, **DISABLE** allows multiple commands to be executed with a history of their result being printed to the screen.

Screen Saver

If enabled, the screen saver will blank out the screen during times of inactivity. The screen saver will be blanked after the specified amount of time has passed with no keystrokes (see *Screen Saver Delay*).

Screen Saver Delay (Minutes)

Sets the number of minutes that the screen saver delays before blanking the screen. The valid range is from 2 to 15 minutes.

Screen Restore on Autohandler

If the programmer is attached to an Autohandler and the screen saver is enabled, this option will keep the screen from being blanked so long as there is ongoing communication with the Autohandler.

Save Configuration

YES writes the present configuration back to disk when you press Enter. The next time you start the program, the same configuration will be recalled. It creates a .CFG file in the same directory as the .EXE that started the program. Therefore, if you delete this file, the software will revert to its original default configuration.

NO allows you to make changes for this programming session only and will not save the present configuration back to disk.

AUTOMATIC saves the configuration to disk every time you exit the program. The program automatically sets all options and reselects the chip when the software is restarted.

The configuration file will be written to the current directory, allowing you to save multiple configurations in different directories. You can override this feature and always use a single configuration file by setting an environment variable to specify in which directory to store that "BP.CFG" file. Place a line in your autohex.bat file similar to this one to use a single configuration:

```
set bpcfg = c:\bp
```

User mode

Selecting **EXPERIENCED** instead of **NOVICE** mode gives you the *Buffer/Clear* command. If programming one device, the program will display the ICC measurement after *Read/Verify* on EPROMs and EEPROMs and after vector testing on PLDs.

Startup messages

The informative messages that appear when the software is started up may be **DISABLED** with this option.

Decimal or Hex

This selection determines whether numeric fields are entered and displayed in decimal (using 0-9) or hex (using 0-9, A-F). This affects the display of the information appearing on the status line at the bottom of the screen.

This is a global command, in that changing this setting within any of the dialog boxes will alter the information displayed in all of the subsequent dialog boxes.

DEVICE COMMANDS

DEVICE/BLANK

Description

Looks for previously programmed locations.

☐ *This command only appears on UV-erasable, One Time Programmable (OTP) and fuse link devices.*

Application

Prior to programming, the command reads a device and determines whether the part is blank. This is a manual command and is often performed automatically under *Device/Options*.

Operation

An EPROM device is verified, making sure all bytes are erased (all 1's or FF on most EPROMs). Any erased part should pass this test. Blank checks are performed at a reduced Vcc level (4.7V), making the EPROM more sensitive to unerased bits.

PLDs are checked by verifying that all of the fuses are intact. Any new PLD should pass this test. Alternatively, you can set the "*Blank check before programming*" option to **ILLEGAL BIT** in *Device/Options* to check only the fuse locations that should not be programmed, allowing you to program on top of partially programmed devices.

EPROM-based parts should be checked before programming. Erasing an EPROM is not as simple as it seems. In order to clear data in an EPROM (in preparation for programming), the chip is exposed to short-wave ultraviolet light, which is typically provided by an EPROM eraser. The UV light penetrates a quartz window on the top of the package, erasing the data. A certain dosage is required to fully erase the part, so brighter light sources erase the part more quickly. Most erasers require between 3 and 30 minutes to erase an EPROM.

Erasing EPROMs

EPROMs are usually erased using a mercury vapor bulb emitting 2537 Angstroms. The bulbs most commonly used are like fluorescent tubes, but without the phosphor; they are often called germicidal bulbs. The ultraviolet light can be harmful to eyes, so erasers are equipped with shields to prevent light leakage.

Erasing an EPROM properly is not as simple as it appears. It is possible to have a partially erased chip that appears to be blank but is not blank when read at a different voltage or temperature. Use this procedure to determine a safe erasing time:

- Start with a programmed chip.
- Erase the part in one minute increments and use the *Device/Blank* command to test it each minute.
- Once the programmer says the chip is blank, double the erase time to give yourself an adequate safety margin.

Some types of parts take longer to erase than others. You may need to experiment with the various parts you use. An EPROM based part with a security bit feature (a PLD or microcontroller) is designed so that the security address will typically be the last bit to erase.

☐ *The adhesive used on labels often blocks UV light. If the chip erases slowly, try cleaning the window with alcohol or a stronger solvent.*

Sunlight and fluorescent light can erase chips; however, it usually takes months or years. You should cover the window with an opaque label to make the data permanent.

Some EPROM based parts are available in inexpensive plastic packages. These parts can't be erased because they have no window. These chips are referred to as one time programmable (OTP) EPROMs.

☐ *EEPROM-based parts do not need this command. EEPROM erasure occurs automatically before programming.*

✎ **When secured, some devices appear blank, and will not program. Most secured devices, however, will appear to be programmed.**

DEVICE/COMPARE

Description

Compare the chip's data to the buffer's data and show any differences on-screen.

Application

This is useful if a verify operation fails and you want to see why.

Operation for memory devices

The default setting reads the chip at its nominal operating voltage (+5.0V). If you are in the EXPERIENCED mode (set under the *Configure* command), a dialog box will appear and allow you to alter the test voltage from +4.0 to +6.5V in 0.1V steps. Once the test voltage has been chosen, if applicable, the data in the memory device is examined and compared to the information in the data buffer. Any differences will show a message similar to the following example:

Data error at 0000: Buffer = 02; Chip = 00

The command does not return an error if there are differences; it just shows the differences. If SET mode is selected under *Device/Options*, you are prompted to insert each chip in the set and press a key to continue.

Operation for PLDs

The part is read at a nominal voltage as specified by the manufacturer's algorithm.

If discrepancies occur, a window will appear revealing that information. Fuses that do not read correctly are highlighted and the cursor is automatically placed on the first error. The values on the screen are the values actually read from the device; therefore, the highlighted bits are the opposite of what was loaded in the buffer. <**Alt-N**> is used to find the next error. <**Alt-P**> finds the previous error.

The current cursor position is displayed at the bottom of the window.

See Also

Device/Options, Device/Verify

DEVICE/CONFIGURE

Description

Configure the special features of a device to operate in a particular mode when placed in the target circuit.

This command appears only on some devices and is specific to the device selected.

Application

Some microcontrollers allow setting of such things as the memory mapping, watchdog timers, security, and clock source.

Operation for special devices

Some devices have special operating modes that can be configured with this command; for example, the PIC microcontrollers configure command looks as follows:

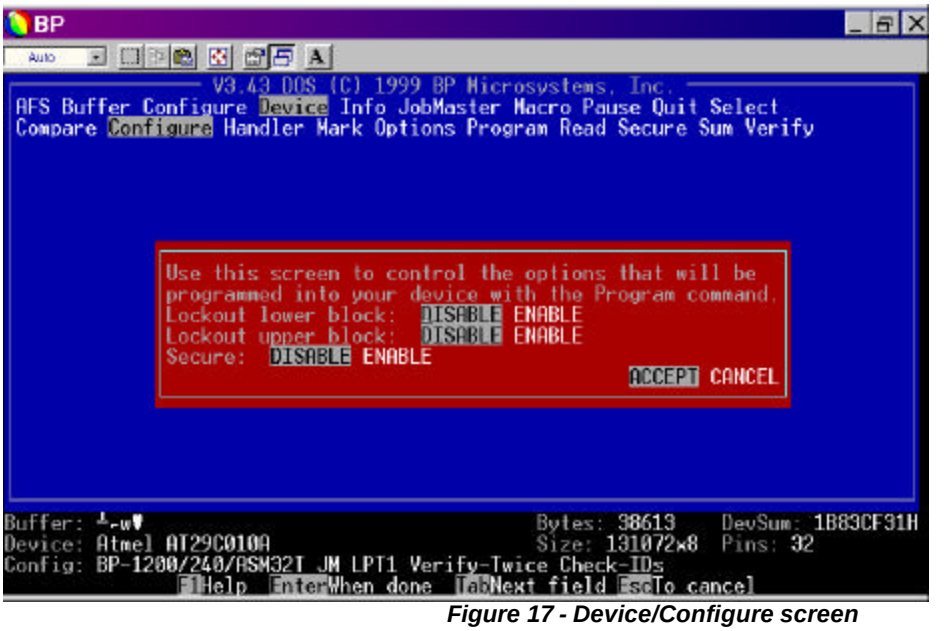


Figure 17 - Device/Configure screen

- Use the **<Tab>** key to select the field you want to change.
- Press the **<F1>** key on any field to get context-sensitive help.
- Press **<Enter>** when finished.

The options in this command will vary from chip to chip; you should refer to the data book for your particular device for more information on the configuration parameters and press **<F1>** for more help on each item in the dialog box. The parameters shown on the screen represent those options that will be programmed into the chip.

See Also

Device/Options, Device/Encrypt

DEVICE/E-FIELD

Description

Enter information to be programmed into the E-Field in devices that support this feature.

Application

This command allows for a user definable fuse pattern that allows for programming the power setting and slew rates for each of the macrocell blocks in the device. This section affects the electrical operation of the device. If an E-Field is programmed, the section of the chip controlled by the E-Field is powered-down when not in use, to reduce total power consumption for the device. The power setting can be set to either "full power" or "low power".

Operation

This command is only available for electrically erasable PLDs, which contain several fuses in the JEDEC map that constitute the device's E-Field. When this command is selected, a dialog box will open allowing you to enter the desired ASCII or hex information (see *Chapter 4, Using the Data Editors*, for information on how to edit this array). The size of the array depends on the number of fuses allocated as the E-Field for the device.

☐ *Editing this array simply changes the appropriate fuses in the current buffer. Thus, you must perform a* Device/Program *in order to update data in the chip.*

Even though you could edit the fuses directly with the *Buffer/Edit* command, it is preferable to use this command because each chip differs in where the E-Field is located in the fuse map and how many fuses may be involved.

See Also

Device/Program, Device/Read

DEVICE/ENCRYPT

Description

Encrypt the program and data in a microcontroller.

☐ *This command appears only on microcontrollers that support data encryption.*

Application

Some microcontrollers (particularly 87C51 families) allow a special level of security that encrypts your program and data by XORing it with an encryption array supplied by you. This allows you to protect your design secrets, yet still be able to verify the contents of devices after they are placed in service. The typical encryption method causes the chip to return all ones, thus making it impossible to guarantee that the contents of the onboard EPROM have not been altered accidentally.

Operation for microcontrollers

This command appears only on some devices and is specific to the device selected.

- Use the **<Tab>** key to select the field you want to change.
- Press the **<F1>** key on any field to get context-sensitive help.
- Change selections using the **<Left>** and **<Right>** direction keys.
- Press **<Enter>** when finished.

Edit the encryption array

When you change the field from **NO** to **YES**, the Encrypt editor will pop up at the bottom of the screen and allow you to insert the desired array values (See *Chapter 4, Using the Data Editors*, for information on how to edit this array). The size of the array may be 32, 64, or 128 bytes, depending on the size of the device, and may even be larger on newer devices.

The array is stored at the bottom of the buffer and will be saved with the file when you do a Buffer/Save*; thus, it will be loaded on a subsequent* Buffer/Load*.* ***This is very important because you will not be able to read the device properly without this array***.

Program Encryption array

If you change this field to **YES**, the current contents of the encryption array will be programmed into the device if and when you accept the changes made. You may choose not to program the encryption array by leaving this as **NO**, or by using **<Esc>** to terminate the dialog box.

Decrypt during Read & Compare

After programming the encryption array, subsequent compare operations will fail unless you change this field to **YES**. If an encrypted device is read without setting the *Decrypt during Read & Compare* field to **YES**, the buffer will contain the encrypted data rather than the actual data in the chip.

- *This will not affect the Verify command; the Verify command does not decrypt data and will fail if the data is encrypted.*

Program specific lock bits

This last field lets you selectively program the lock bits. For comparison, the *Device/Secure* command will always program all bits for maximum security. Refer to your device data book for more details on what the lock bits do.

See Also

Chapter 4, Using the Data Editors, page 4-1

Device/Configure; Device/Secure

DEVICE/HANDLER

Description

Operates the Handler, if an autohandler is present, programmer in a production programming mode, with or without an Autohandler (automated device Handler).

Application

Prior to using the programmer in a high-volume production setting, select which device Handler will be used.

- Manual *mode is a convenient way to keep track of the number of successful and failed programming attempts without actually using an Autohandler.*

Operation

A dialog box will appear showing the current settings for the Handler options.

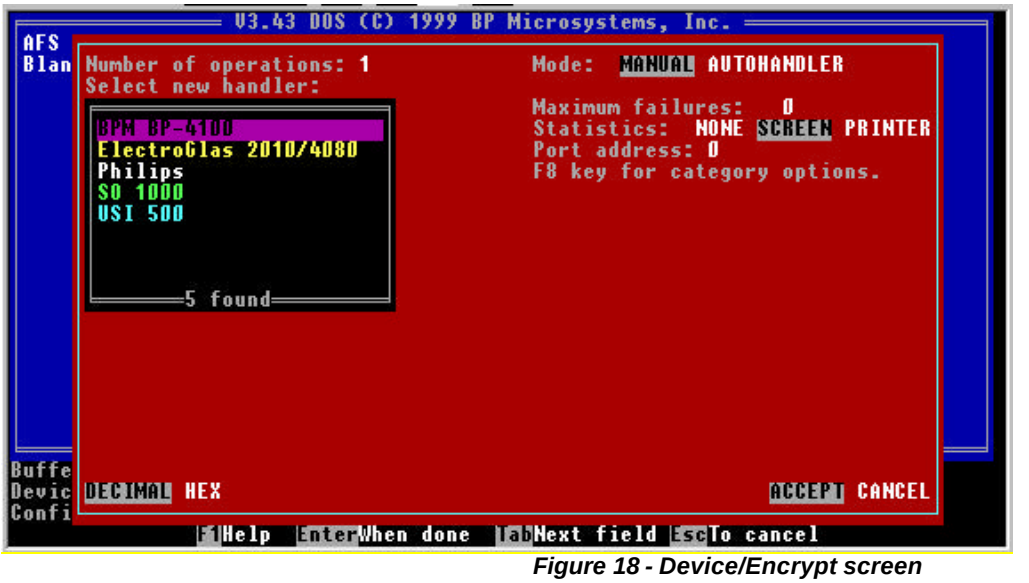


Figure 18 - Device/Encrypt screen

- Use the **<Tab>** key to select the field you want to change.
- Press the **<F1>** key on any field to get context-sensitive help.
- Change selections using the **<Left>** and **<Right>** direction keys.
- Press **<Enter>** when finished.

Number of operations

Specify the number of operations for this run. This number is typically the number of chips you wish to program. However, if you are using the SET programming mode (see *Device/Options*) with more than one chip per set, then the number of operations must equal the number of chips divided by the number of chips programmed for each set.

This combination is not usually recommended.

Mode

MANUAL is the default setting and is intended to simplify programming numerous chips with the same data. It will keep the same statistics that are kept when in the **AUTOHANDLER** mode and will display the progress at the bottom of the screen. This mode prompts you when to insert the next chip. After the next chip is inserted, simply hit any key to perform the operation again and again.

The **AUTOHANDLER** mode should be selected when you wish to have the software orchestrate the operation of the Handler and the programmer. In this mode, all you have to do is feed the chips to the Handler.

Select new Handler

Type in a Handler name or use the arrow keys to select the Handler you are using.

Autohandler software support is not a standard feature of the BP-4100/4500 and must be purchased from BP Microsystems. Call us for information on BP Microsystems' Advanced Feature Software (AFS) support, which may be field installed (contact information is listed in Chapter 12, Troubleshooting and Maintenance, page 14-1).

Maximum failures

Specify the number of acceptable failures for this run. If you are in the AUTOHANDLER mode, the software will abort this job when this number is exceeded and prompt you on it.

Statistics

The default is to send the statistics to the **SCREEN**. However, you may send them to the **PRINTER** instead. To be able to print to a printer you will need at least two parallel ports in your computer and possibly three, if you are using a parallel interface to the Handler.

Port address

This is the address of the parallel, serial or custom port that is used to communicate with the Handler. It is not necessary to set this address if you are using the **MANUAL** mode (see "*Mode:*" above). The Autohandler normally connects to a second parallel port (it cannot use the same port as the programmer). The port address will be set automatically if you have only two ports. Machines with three LPT (parallel) ports may require manual input to change this port number.

Category codes

The category codes are used to specify the output bin or tube for particular error conditions or successful operations. Most autohandlers are set up where category code 1 is used to indicate a good chip and category code 2 is used for all bad chips. Handlers with more than two output bins usually allocate a single bin for bad devices and the remaining bins for good devices. Therefore, the default values should work on most Handlers. If you wish to characterize the device failures further, you may specify different category codes for specific failures provided the Handler is configured correctly to accept the additional codes.

The most common use of extra category codes is to sort out continuity test failures so the parts can be examined for bent pins, reverse insertion, etc.

You should refer to the Handler documentation to determine and set the interpretation of these category codes.

Example: *Exatron Handlers may have a hardwired binning configuration in place when using the parallel interface. In. fact, on an 8-bin Handler, Exatron Handlers will probably map bin 2 codes to the output directly under the DUT (device under test) and bin 1 codes to all other outputs. If you use the serial interface to the Exatron Handlers, the category codes are single ASCII characters that will be sent to the Autohandler. On MCT Handlers, you can configure the category codes received to sort the device to any output by setting the matrix of DIP switches.*

We recommend placing your output bin for defective devices directly below the DUT site. This is done to guarantee that a bad device is never passed over the top of a good tube, in case it is accidentally dropped due to a power failure, jam, or some other unforeseen condition.

DEVICE/MARK

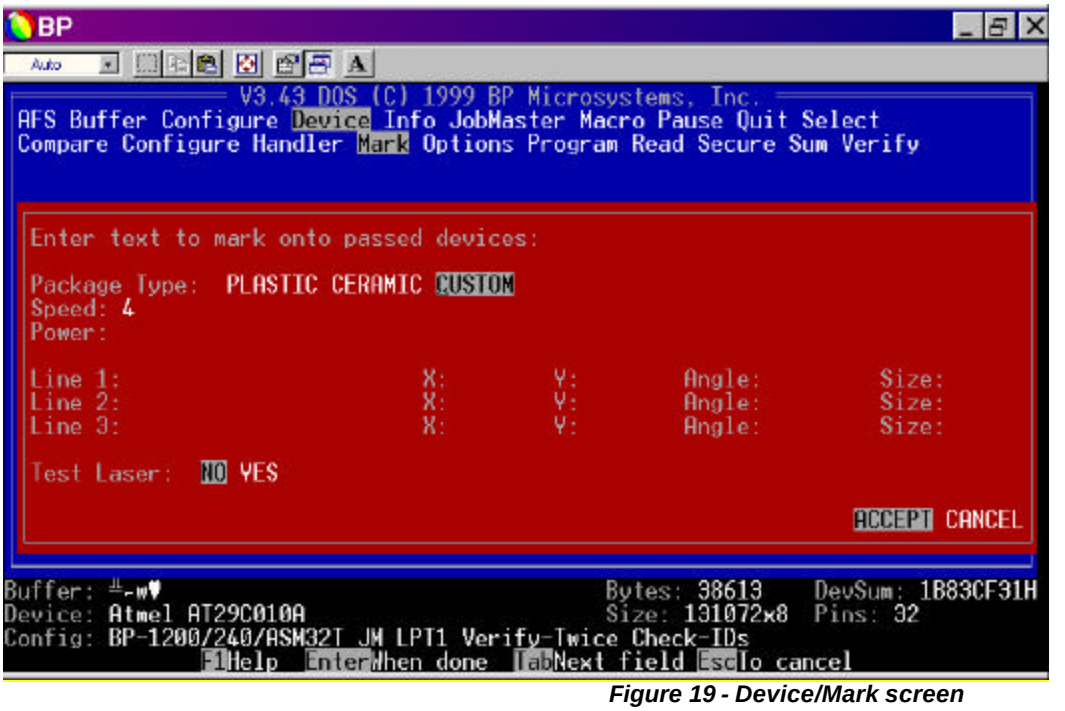


Figure 19 - Device/Mark screen

Description

Enter text information for lasing or labeling.

Application

Use the Device/Mark option to enter information for lasing or labeling devices.

Operation

In the *Device/Mark* screen, choose a package type: plastic, ceramic, or custom. When choosing plastic or ceramic, you will need to specify *Text, X-axis, Y-axis, Angle* and *Size.* When choosing custom, you also need to specify *Speed*.

DEVICE/OPTIONS

Description

Allows you to specify different programming, reading, and testing options to reflect your target system requirements.

☐ *This should be done prior to performing a* Device/Program *command.*

Hot-Key **Alt-O**

Application

Use these options to:

- program multiple chips from a single file
- split data for wide data paths (16 bits or larger) and multiple EPROMs in sequence
- select the byte order when using 16-bit wide chips
- specify range and buffer offset parameters
- control command execution such as enabling/disabling continuity tests, electronic identifiers, blank checks, verify passes, test vectors and securing
- set the interpretation of test vector X-values.

Operation for memory devices

You may or may not see all the options below, depending on the programming characteristics of the chip you have selected. Below is an example of the options for a 16-bit wide EPROM:

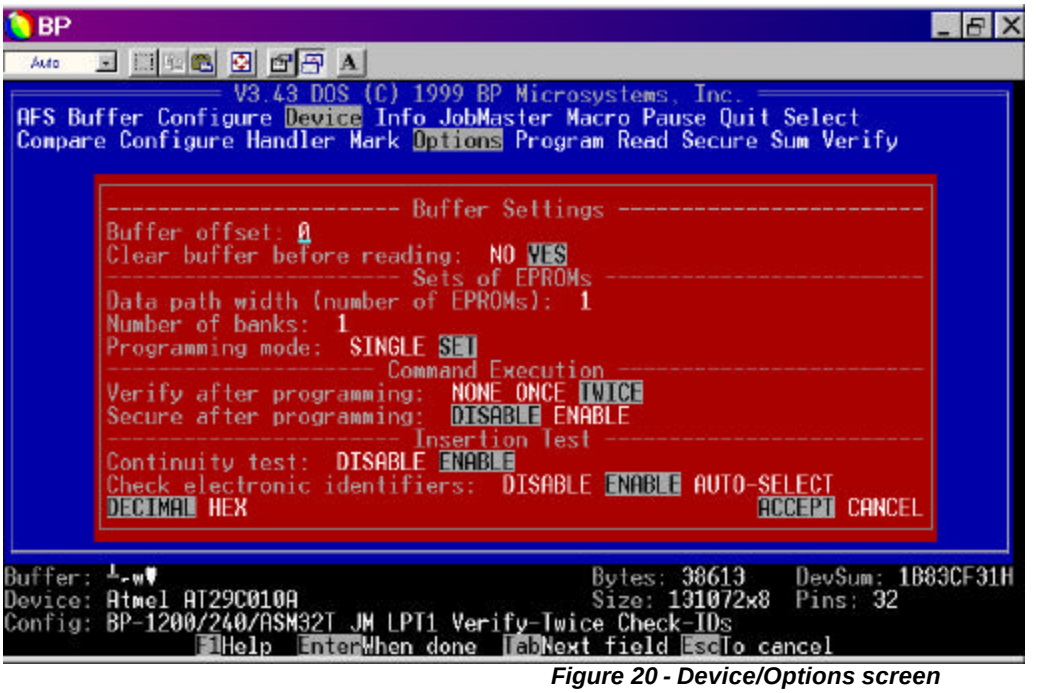


Figure 20 - Device/Options screen

- Use the **<Tab>** key to select the field you want to change.
- Press the **<F1>** key on any field to get context-sensitive help.
- Change selections using the **<Left>** and **<Right>** direction keys.
- Press **<Enter>** when finished.

Autostart on Continuity

When enabled, this option will prompt the programmer to verify continuity on all pins of a device prior to beginning a device operation.

Starting word of range

This option allows you to define the starting physical address within the memory device that you are programming. This is handy for programming or reading only part of a chip, or for placing data in specific addresses. The minimum parameter for the starting word of range is "0". This parameter is used in conjunction with the *Ending word of range* defined below.

Ending word of range

The *Ending word of range* setting is used with *Starting word of range*, explained above. The maximum end address parameter will change based upon the part you are programming - the larger the device the higher the address. A 256-Kilobit EPROM would have 32,767 as its highest writable address. An 8-bit wide 1-megabit chip

would have 131,071 as the highest address. You may not set this option to a number less than the value specified in the *Starting word of range.*

The above range options also allow the use of 16-bit wide addresses; however, you cannot specify a byte address to program. The highest address for a 16-bit wide part will be one-half that of a similar 8-bit.

Example: *A 1-megabit 16-bit wide device would have 65,534 as its highest address.*

- *Some EEPROMs and Flash memory devices require erasing the entire address space or large sectors, and will not support range programming. Thus, these options will not appear on all devices.*

- *Some parts have extra configuration bytes that reside at the end of the normal address space. If this is the case, then you can not specify a range within these bytes, i.e. – the starting address must be below or equal to the first address of the configuration bytes and the ending address must be below the first configuration address or equal to the last address.*

Buffer offset

This option allows you to program or read a chip from or into any address within the data buffer. Using the *Starting word of range*, *Ending word of range* and *Buffer offset,* you have the capability to program/read data into/from any address on the chip, and from/into any address in the buffer. When *Buffer offset* is negative, the first byte in the buffer corresponds to a point in the middle of the chip specified by the offset. When *Buffer offset* is positive, the first byte in the buffer is not used; the first byte in the memory corresponds to the specified byte in the buffer. If a negative value is specified for *Buffer offset,* a positive value must be specified for *Starting word of range*. The default is zero.

Example: *You want to program the second half of a 27256 (8000 hex bytes in size) using 4000 bytes loaded into the buffer. Set both the* Starting word of range: *and* Buffer offset: *to -4000 before programming so the data at address 0 in the buffer will be programmed into the device address 4000, so only the second half of the EPROM will be programmed.*

You want to program a 27256 using data that is loaded into the buffer at address F0000. Set the Buffer offset: *to F0000. This will tell the programmer to read buffer address F0000 to program EPROM byte 0 and so forth. An alternative (and faster) way to accomplish the same task is to specify the buffer offset when the file is loaded; see* Buffer/Load *command.*

Clear Buffer before Reading

YES permits you to clear the data buffer prior to reading information from a chip. **NO** does not clear the buffer data. A read will then only alter the addresses specified by the *Starting word of range,* the *Ending word of range* and the *Buffer offset.* This is very useful if you wish to inspect a small amount of data in a chip, or you wish to combine data from multiple chips into the buffer.

Byte order

For 16-bit wide memory device systems using a byte-reversed format, such as 16-bit wide EPROMs used in conjunction with Intel processors, use the first setting, **MSB=1, LSB=0**. In this mode, the **MSB** (Most Significant Byte), outputs 15-8 on the chip, contains bytes with odd addresses (1, 3, 5...). The **LSB** (Least Significant Byte), outputs 7-0, contains even addresses (0, 2, 4...). Systems using a non-byte-reversed format, such as Motorola microprocessors, use the second setting, **MSB=0, LSB=1**, where the **MSB** is even addresses (0, 2, 4...) and the **LSB** is odd addresses (1, 3, 5...).

Data path width (number of EPROMs)

The data path width is the number of EPROMs read simultaneously in the target system.

Example: *If you are programming pairs of 8-bit chips for a 16-bit microprocessor, set it to 2. Likewise, if you are programming four 16-bit wide EPROMs for a 64-bit wide system, set it to 4. These groups of EPROMs are referred to as a single bank.*

Number of banks

If the data path is set to 1, then the number of banks is simply the number of EPROMs you want to program (*i.e.*, the address space divided by the number of EPROMs). When you program a file that is larger than one bank of EPROMs, you set the number of banks here.

Example: *If your file is 512K bits and you are using 128K bit EPROMs, then you will need 4 banks of EPROMs to hold all the data. When you program pairs of EPROMs (data path width = 2), the number of banks is the number of pairs required, not the total number of chips.*

Programming mode

When you program multiple EPROMs in a set (*i.e.*, width or number of banks is greater than one), SET mode will prompt you to insert the EPROMs one at a time, in order. If you want a specific EPROM out of the entire program set (such as in a production run or when replacing a single blown-out chip), use the SINGLE mode. Another dialog box appears after you press **<Enter>**. The second box lets you enter

which EPROM to program or read. The desired EPROM position is described as the data position within a bank and the bank number. Both numbers start counting at 0. For a 16-bit system with 8-bit chips, 0 selects even addresses and 1 selects odd addresses.

Erase before Programming

When enabled, this option will cause the programmer to perform an erase cycle to be performed on a device. If all devices are known to be blank, this option can be disabled to allow for faster throughput.

Blank check before programming

This option will force the programmer to perform a blank check before programming, ensuring that you have not placed a partially erased or fully programmed part in the programmer site. The blank check is performed at a lower Vcc voltage, to ensure complete erasure.

Verify after programming

A verify operation is performed during programming on almost every memory device, so this verification step is optional. The preferred setting is TWICE, verifying EPROM contents with Vcc at two different voltage extremes, such as 4.7V and 5.3V. Passing the test is a good indicator that the part is fully programmed, properly erased before programming, reliable over a wide range of operating temperatures, and will retain data during its operational life. Performing verification ONCE (at 5.0V) is quicker, but less exhaustive. Disabling the verification is faster and a good choice for prototyping.

Continuity test

Leaving this **ENABLED** verifies the chip is inserted correctly in the programmer site. If not correctly inserted, an error message is displayed. **DISABLE** prevents this test from occurring.

The continuity test will tell you exactly which pin drivers on the programmer are not connected to the chip's internal die. If almost all pins are disconnected, then one or more ground pins are probably blown out. One or a small number of pins without continuity could be caused by something as simple as the pin on the chip or the programmer being dirty. If cleaning the pins does not help, then the bonding wire or the output pad of the die is probably damaged.

- **There are some rare cases where the continuity check cannot be performed because the chip manufacturer specifies not to or because doing so would adversely affect the device. In these cases, it is possible that a continuity problem will go undetected.**

Check electronic identifiers

Electronic identifiers are codes that can be read from a chip to identify its manufacturer and programming algorithm. Almost all of the EPROMs produced today have this feature. Selecting **ENABLE** will reach the chip code when you program it and verify the chip against the expected code. If the code does not match, you will get an error message and be given the choice to abort the operation, retry it with a different chip, ignore the error, or select a different chip. Choosing **AUTOMATIC** will select the correct device and skip the error message automatically. If it fails to find the device code of your chip, you will see an error message like the one mentioned above. Selecting **DISABLE** is useful if the software expects an identifier code for your chip, but your chip does not have one. This happens when a manufacturer has been making the chip without the ID code, then adds the ID code in a later manufacturing process. Placing an older chip in the programmer site will cause an error since it doesn't have the code. Choosing **DISABLE** will prevent an error message in this case.

Operation for PLDs

Below is an example of the *Device/Options* available when you have selected a PLD:

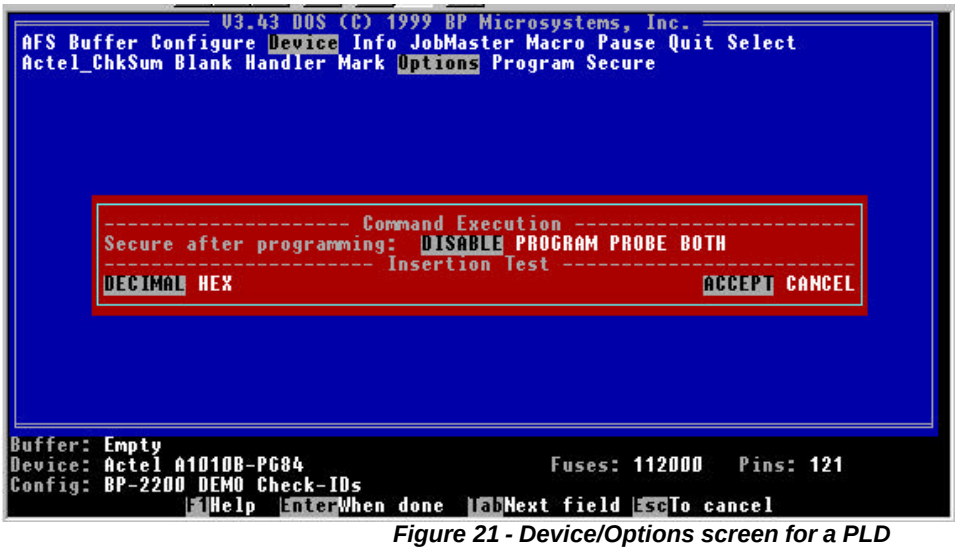


Figure 21 - Device/Options screen for a PLD

- Use the **<Tab>** key to select the field you want to change.
- Press the **<F1>** key on any field to get context-sensitive help.
- Change selections using the **<Left>** and **<Right>** direction keys.
- Press **<Enter>** when finished.

Display

When a vector test is performed with the *Device/Test* command, you can choose to display:

- only error messages (ERRORS-ONLY is the default);
- all vectors and error messages (ALL); or
- vectors and error messages, displayed one at a time (SINGLE-STEP).

X-value

When a set of test vectors is generated, pins marked "X" indicate a "don't care" condition. This condition is treated differently by different people, equipment and software. BP Microsystems' programmers provide several options to maintain the highest compatibility:

☐ *If "X" is applied to both input and output pins in your JEDEC file, then you must use the PULL-UP setting. The pin will then use a 5k pull-up resistor. The "X" value is also set from the JEDEC file if defined there. If you don't use the PULL-UP setting, the programmer attempts to force the chip's pin to either a high or low state.*

✎ **Forcing a low output to a high state can permanently damage the chip! This may cause an excessive current error to occur.**

You can choose to have "X" inputs tied low or high with the 0 and 1 settings. Selecting **RANDOM** or **PSEUDORANDOM** sets chip pins either high or low at random. A random number generator originates a sequence of bits based on a seed value. Choosing **RANDOM** generates a seed value at random. Choosing **PSEUDORANDOM** will not generate new seeds, so the test sequence will be the same every time. You can change the sequence by changing the seed.

☐ *The "*Seed:*" field will only appear when you have selected the PSEUDORANDOM choice.*

Vector test after verify

You can choose to perform a vector test automatically after verifying, when test vectors are available. Choosing **TWICE** will use both low and high Vcc voltages and **ONCE** will only use a nominal voltage.

Vector test after secure

You can choose to perform a vector test automatically after securing, when test vectors are available. This is useful to ensure that the secure operation did not affect

the functionality of the device. Choosing **TWICE** will use both low and high Vcc voltages and **ONCE** will only use a nominal voltage.

Blank check before programming

This option will force the programmer to perform a blank check before programming, ensuring that you have not placed a partially erased or fully programmed part in the programmer site.

The **ILLEGAL BIT** option is useful when you only want to make sure that all unprogrammed locations in the fuse pattern are not programmed on the chip. It will ignore locations that you intend to program, allowing you to program a fuse pattern on top of a partially programmed chip.

Secure after programming

Certain chips, such as PLDs, microcontrollers, and write-protectable NVRAMs, have a secure feature. The part will continue to operate normally, except that the EPROM or fuse data cannot be read out (except for NVRAMs). Selecting **ENABLE** will automatically secure the part, after programming and verifying. See *Device/Secure* for more information.

Continuity test

Leaving this **ENABLED** verifies the chip is inserted correctly in the programmer site. If not correctly inserted, an error message is displayed. **DISABLE** prevents this test from occurring.

The continuity test will tell you exactly which pins on the chip are not connected to the internal die. If you get almost all pins disconnected, then one or more ground pins are probably blown out. One or a small number of pins without continuity could be caused by something as simple as the pin on the chip or the socket module being dirty. If cleaning the pin does not help, then the bonding wire or the output pad of the die is probably damaged.

There are some rare cases where the continuity check cannot be performed because the chip manufacturer specifies not to, or because doing so would adversely affect the device. In these cases, it is possible that a continuity problem will go undetected.

See Also

Device/Blank, Device/Program, Device/Read, Device/Verify, Device/Secure, Device/Test

DEVICE/PROGRAM

Description

Program a device from data in the buffer.

Hot-key **Alt-P**

Application

To permanently store your data or design in a chip.

Operation for memory devices

If the part is electrically erasable (EEPROM based) and the *Device/Option* **ERASE** is enabled, it is automatically erased before programming. If not, the part must be blank before programming.

The device will be blank-checked if that option is enabled with the *Device/Options* command. The optional two-voltage verify step will guarantee with high confidence that the part is programmed and operating correctly.

- *If you have selected a SET programming option with* Device/Options*, you will be successively prompted to insert each chip to be programmed. If you have selected only a single chip, no prompt will appear.*

Operation for PLDs

The device is erased if possible or checked to see if it is blank (when enabled in *Device/Options*). The device is programmed and verified. Some parts will be verified at two Vcc voltages and others only at one voltage, as specified by the semiconductor manufacturer. Parts producing errors while programming or verifying should be considered defective. Bipolar parts that fail to program can often be returned to the manufacturer for replacement if they have not been secured. See *Chapter 10, Troubleshooting and Maintenance,* for more information.

A vector test is automatically performed if this option is enabled under the *Device/Options* command and test vectors are loaded into the buffer. The vector test is very important on fuse-link programmable parts.

Finally, the part may be automatically secured if this option is set under the *Device/Options* command. This step will not execute if the device fails to verify or vector test properly. You may also vector test the device after securing it by setting this option under the *Device/Options* command.

See Also

Chapter 10, Troubleshooting and Maintenance, page 14-1.

Buffer/Load, Device/Options, Device/Read, Device/Handler

DEVICE/READ

Description

Place chip contents into a data buffer.

Hot-key **Alt-R**

Application

Used to interrogate the contents of a chip for identification, copying or modification.

Operation

The appropriate buffer is cleared, the device is read, and its data is stored in the buffer. The checksum will be displayed on the buffer status line. The buffer may be edited, saved to disk, or used to duplicate the chip.

- *If you are reading EPROMs and you have set one or more SET programming options with the* Device/Options *command, you will be successively prompted to insert each chip to be read. If you select a single chip only, no prompt appears. If you are in the SINGLE mode and there are multiple chips in a set, the buffer will not be cleared before reading the chip.*

- *PLD test vectors are not stored in a PLD, so they cannot be read. The test vector buffer will be empty after reading the chip.*

- *Devices that have been secured cannot be read properly. Secured devices may appear all blank, fully programmed, or scrambled. The exception is write-protected NVRAMs and EEPROMs. See the* Device/Secure *command for more information.*

After reading the chip contents you may view or edit it with the *Buffer/Edit* command, save it to disk with the *Buffer/Save* command, or program it into another chip with the *Device/Program* command.

When using multiple operations in the *Device/Handler* command, this command will not repeat since reading the subsequent parts will overwrite the data read by previous parts.

See Also

Buffer/Edit, Buffer/Save, Device/Options, Device/Program, Device/Handler

DEVICE/SECURE

Description

Secure a PLD or a microcontroller so it cannot be read, or write-protect an EEPROM or NVRAM so it cannot be written in the target circuit.

This command only appears when you have a PLD, micro, or write-protectable device selected that actually has the ability to be secured.

Application

To prevent unauthorized duplication of a proprietary design or to prevent accidental writing to an in-circuit programmable device.

Operation for PLDs and microcontrollers

This command appears only when the selected device can be secured. Some microcontrollers and PLDs can be secured by programming a special location. When the device has been secured, it cannot be read correctly, verified, or duplicated.

Most bipolar PLDs can be secured by blowing a security fuse link or some other non-reversible means. Once a PLD is secured, it cannot be read properly, but will function normally (*i.e.*, still pass test vectors).

EPROM and EEPROM based PLDs generally can be secured by programming a certain location. The security bit will be cleared when the part is erased by the *Device/Program* command.

Typically, on a UV erasable PLD or microcontroller, the security bit address is designed to erase last. Thus, a secured part may take longer to erase.

Operation for memory devices

Some NVRAMs and EEPROMs have a write-protect feature that prevents unintentional writing to the part. Thus, executing the *Device/Secure* command on these devices does not actually disable the ability to read the contents of the chip (which would make the chip useless in a circuit). It sets the chip up such that the device requires a sequence of software commands (that are extremely unlikely to occur accidentally) to disable this protected mode.

For devices that may have already had this write-protection performed by our programmer or your circuit, the Device/Program *command automatically unprotects the device prior to writing the buffer contents to it.*

See Also

Device/Program, Device/Read, Device/Verify

DEVICE/SUM

Description

The *Device/Sum* command reads data from a chip and calculates the checksum without altering the data in the buffer.

- *This command is available only on memory devices.*

Application

This command allows a programmed chip checksum to be verified against a file checksum. The *Device/Sum* command may be used to compare checksums without erasing or destroying the data in the chip. The checksum will be printed on the screen; no error message will result if the checksum is not correct (that is for you to decide).

- *The* Device/Sum *command displays a four digit hex sum which should be the same as the four least significant digits of the buffer's displayed sum (except as noted above). The reason the sum is displayed as four digits is because the programmer can calculate the sum using 16 bit arithmetic much faster than using 32 bit operations.*

Operation for memory devices

Place a chip containing programmed data into the programmer site. Use the *Device/Sum* command to calculate and display the checksum of the device (it will match the buffer checksum used to program the part, as displayed on the status area of the display, only if the buffer default value was 0 and the entire buffer was programmed into the part. Any bytes in the buffer at addresses higher than the memory device size will not be reflected in the device sum).

See Also

Device/Read, Buffer/Load, Device/Verify

DEVICE/TEST

Description

Performs a vector test on a PLD.

Hot-Key **Alt-T**

Application

Verify that a programmed chip is functioning correctly by applying test vectors. Some PLDs may not be guaranteed to operate as intended after programming and verification. Properly designed test vectors can verify that the part is fully functional.

Operation for PLDs

The device is tested in its normal operating mode. The programmer applies high and low logic voltages to the part's inputs and its outputs are verified against expected values.

The command output depends on the *Device/Options "Display"* settings:

- **ERRORS-ONLY**: No output will be generated unless errors are encountered.
- **ALL**: Each vector will be displayed as it is used to test the chip.
- **SINGLE-STEP**: Each vector is displayed before testing. The part is tested and the test resultant vector is displayed on the following line. Press any key to see the next test vector.

Errors are indicated by showing the vector causing the error, the test resultant vector, and a message on each pin failing to verify.

Test resultant vectors use the same format as ordinary test vectors. The character for each output pin (specified by **H**, **L** or **Z**) is replaced by the result at that pin. If the result is an error, a message will be printed below. Pins labeled **X** will be displayed as **0**, **1**, or **Z**, depending on the "**X**" option selected under the *Device/Options* command.

See Also

Device/Options, Device/Program, Device/Verify

Test Vectors in Chapter 10 – Troubleshooting and Maintenance, page 14-1.

DEVICE/UES

Description

Edit the User Electronic Signature in some PLDs.

Application

Some electrically erasable PLDs have several fuses in the JEDEC map, which are designated as the User Electronic Signature or UES. These locations do no affect the operation of the device and may be set to any value useful to you for identification purposes, such as the printed circuit board position or the date and revision of the logic.

Operation for PLDs

This command appears only on some devices and is specific to the device selected.

Example: *The Lattice GAL22V10 D looks as follows:*

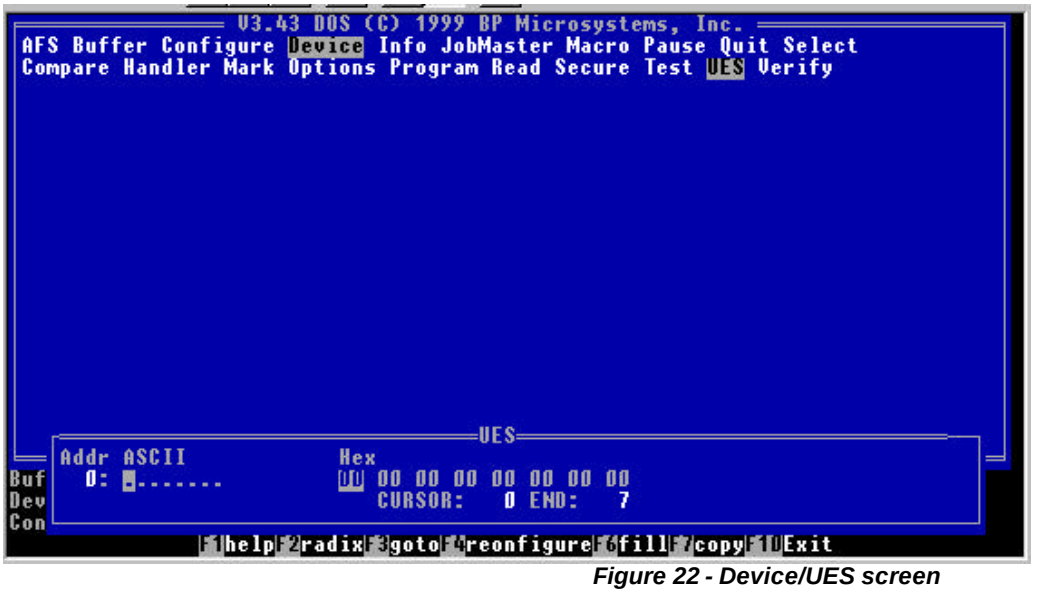


Figure 22 - Device/UES screen

The UES editor will pop up at the bottom of the screen and allow you to insert the desired ASCII or hex values (see *Chapter 4, Using the Data Editors, page 4-1*, for information on how to edit this array). The size of the array depends on the number of fuses allocated as the UES for the device.

Editing this array simply changes the appropriate fuses in the current buffer. Thus, you must perform a *Device/Program* in order to get this data into the chip.

To inspect the UES of an unknown chip, you must first read the chip with the *Device/Read* command and then use the *Device/UES* command to view its contents.

Even though you could edit the fuses directly with the *Buffer/Edit* command, it is preferable to use this command because each chip differs in where the UES is located in the fuse map and how many fuses may be part of the signature.

- **Different chip manufacturers have opposing opinions about whether these UES fuses should be looked at during verify and compare operations. Since we perform the verify operation according to the manufacturer's specifications, some devices will test every fuse in the part, leading to verify errors if the part is functionally identical but the UES does not match exactly. Other parts will not verify the UES so only functional differences will cause a verify or compare error.**

See Also

Device/Program, Device/Read

DEVICE/U-FIELD

Description

Enter information to be programmed into the U-Field in devices that support this feature.

Application

Some electrically erasable PLDs have several fuses, in the JEDEC map, which constitute the device's U-Field. This is a user definable fuse pattern that allows for programming an identification tag into the part. The data can be a serial number, date and time, or any other useful identifying mark. This does not affect the functionality of the device.

Operation

This command appears only on some devices and is specific to the device selected. When this command is selected, a dialog box will open allowing you to enter the desired ASCII or hex information (see *Chapter 4, Using the Data Editors, page 4-1*, for information on how to edit this array). The size of the array depends on the number of fuses allocated as the U-Field for the device.

Editing this array simply changes the appropriate fuses in the current buffer. Thus, you must perform a *Device/Program* in order to get this data into the chip.

Even though you could edit the fuses directly with the *Buffer/Edit* command, it is preferable to use this command because each chip differs in where the U-Field is located in the fuse map and how many fuses may be involved.

See Also

Device/Program, Device/Read

DEVICE/VERIFY

Description

Verify a chip's data is the same as the data in buffer.

Application

Identify data contents of an unlabeled (or unknown) programmed chip, or see if a part is programmed correctly.

Operation

The device contents are read and compared to the buffer contents. If a difference is detected, a single error message will result, indicating that there is a difference. This command is faster than the *Device/Compare* command because it does not keep track of failed locations, it simply halts when the first discrepancy is found.

- *If you are using EPROMs, the* Device/Options *command can force an automatic verification after programming. If you have selected to verify TWICE with the* Device/Options *command, the device will be verified at high and low Vcc limits per device manufacturer specifications. This test confirms that the part will work correctly over a range of power supply voltages and temperatures, and will retain data for its operational life. If you have specified ONCE, the part will be verified once (at a nominal operating voltage) after programming. If NONE was specified, then the part is only verified on a per word basis during programming.*

- *PLDs verify according to the chip manufacturer's proprietary specifications, which may not be altered by the user in any way.*

See Also

Device/Options, Device/Program, Device/Compare

INFO COMMANDS

INFO/BBS

Description

Display the latest information on the BBS lines available from BP Microsystems.

Application

Displays instructions that tell you how to log onto our electronic Bulletin Board System to obtain software updates.

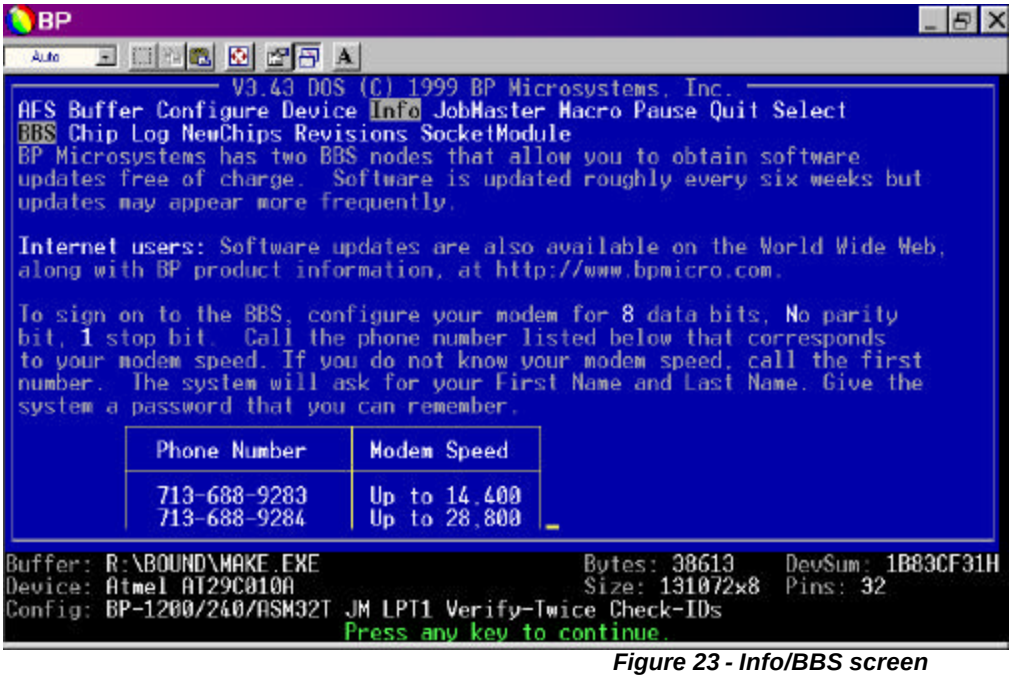


Figure 23 - Info/BBS screen

Press <**Enter**> and read the screen contents. You should get something similar to the screen above. There may be more than one screen of information, so press any key until finished.

See Also

Info/NewChips, Info/Revisions

INFO/CHIP

Description

Display some characteristic information on the currently selected chip.

Hot-keys **F2 or Alt-I**

Application

To provide you with an on-line database of summary programming information on specific chips. This is a useful way to learn which package types are available for a chip and which programmers support a chip.

Special notes regarding the programming or operation of the chip are also found under this command. This command may also be invoked by one of the above hot-keys while on the highlighted chip in the Select *command. After the information appears for the highlighted chip, you may then use the <**Up**> and <**Down**> direction keys to move through the list of chips and continue to get the information for each chip in the list.*

Operation for memory devices

Below is an example of an Intel 27C010A, which happens to have a short programming pulse width of only 10µs. If you multiply the pulse width by the number of words in the device, you will calculate the theoretical programming time, excluding the time required to change addresses and strobe the control pins. The actual programming time will be somewhat slower because the programmer must have some time to complete necessary overhead operations and also because the programmer may be waiting for data and instructions from your PC.

BP Microsystems uses programming algorithms, voltages, waveforms, and times as specified by the semiconductor manufacturer.

- **Users can NOT change these parameters.**

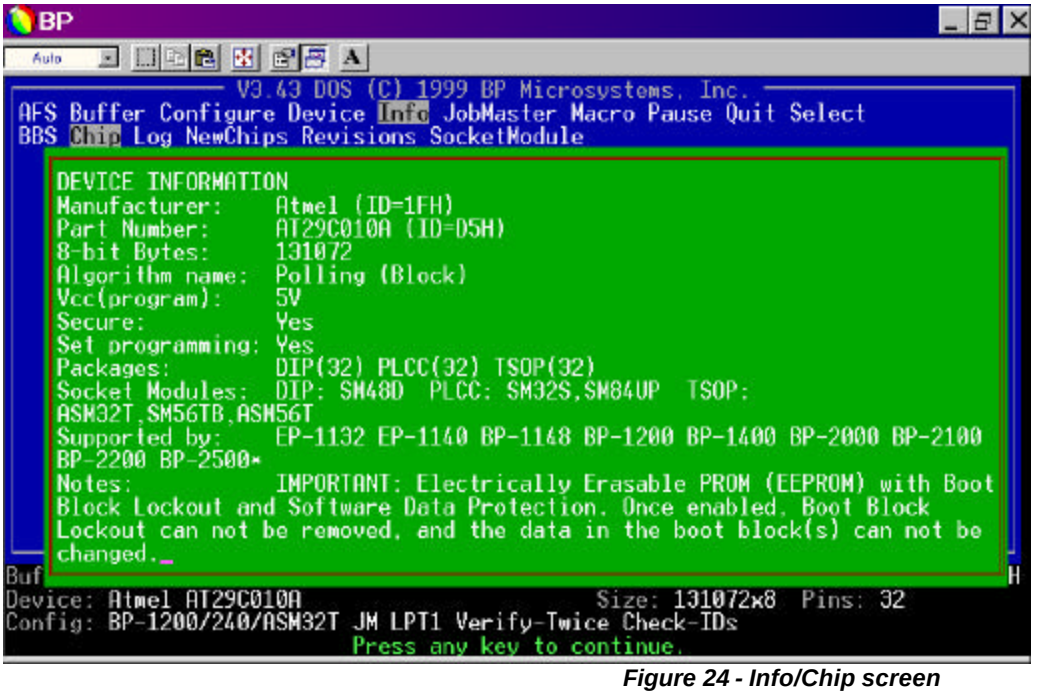


Figure 24 - Info/Chip screen

Most memory devices will be displayed similar to above. However, some will also show a programming overpulse which gets applied on some programming algorithms. All the information shown can be found in the manufacturer's data book for the particular device.

The *Notes* field is where special information about the device goes, such as the necessary programming adapters or the extra configuration bytes in the device.

Operation for PLDs

With respect to the programming characteristics, the information displayed for PLDs is not as informative as for memory devices. The PLD algorithms are proprietary to the chip manufacturers and are not published in general data books. Thus, we cannot disclose any details regarding the programming information, such as voltages, slew rates, or pulse-widths. However, we do display other characteristics that are evident in the data books, such as:

- number of pins and fuses
- whether or not the device is securable, has a register preload feature, and is electrically erasable
- which speed suffixes and package types the device is available in

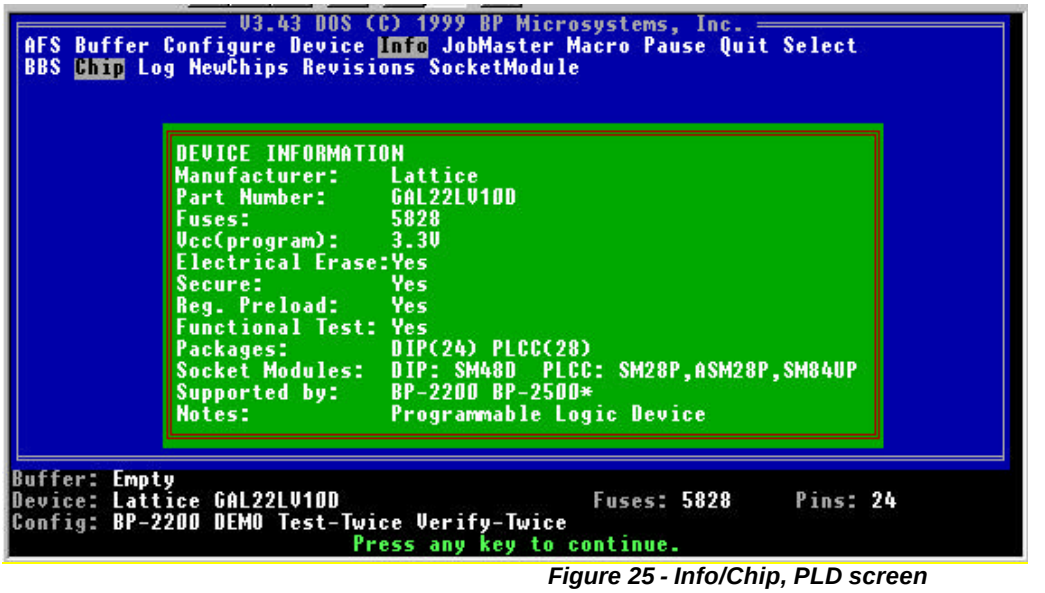


Figure 25 - Info/Chip, PLD screen

The above illustrates an example of a chip for which a special note appears when you use the *Info/Chip* feature.

See Also

Select

INFO/LOG

Description

Send information displayed in main window to a log file.

Application

You may create a log of all the activity during a particular programming session. This log file may be viewed at any time, printed, or saved to another file name. This feature is useful when you are debugging vector errors on a PLD or comparing errors on a memory device. It is also a convenient way to monitor the operation of a person using the programmer in a production environment.

Operation

- Use the **<Tab>** key to select the field you want to change.
- Press the **<F1>** key on any field to get context-sensitive help.
- Change selections using the Left and Right direction keys.
- Press **<Enter>** when finished.

Display log file

NO allows you to choose one of the other options. **VIEW** will disable all other options and display the contents of the current log file in the main window, allowing you to press any key to see the next full screen. **PRINT** disables the "*Clear log file:*" and "*Write to log file*:" fields and enables the "*Send log file output to:*" field (see below). It will let you print to the default printer or to a file name specified by you.

Send log file output to

This option appears only when **PRINT** is selected in the above "*Display log file*:" field. It allows you to print to the default printer port or specify your own printer port or file name. Simply hit Enter if you want to send it to the **LPT** port shown, or <**Esc**> to cancel.

Clear log file

This option allows you to clear the current contents of the log file. It is important to do this periodically; otherwise, the size of the file will get so large that it will use up a substantial amount of disk space.

Write to log file

ON simply opens the current log file and appends all subsequent writes to the main window to the opened log file. The file will grow until this option is set back to **OFF**. The file is not cleared until you do it with the above "*Clear log file:*" option. Thus, you can record some (ON), stop (OFF), record some more (ON), stop (OFF), ...etc., until you have all the information you are interested in saving. At this point you will probably want to print the log file and then clear it for future use.

☐ *This log file is written in the current directory and is named BP.LOG.*

See Also

Device/Handler

INFO/NEWCHIPS

Description

See which new chips have been added recently.

Application

This command is used for information only. It lets you know what chips were added on particular revisions of the software.

Operation

Just type in the range of the software revisions you wish to inspect and the results will be displayed in the main window. The chip manufacturer, part number, and which programmer models support that part will be displayed.

- *When you log into BP Microsystems' BBS (see* Info/BBS*), you may inspect a merged version of this list and the revisions generated by the* Info/Revisions *command. Choose the* Bulletin *command from the main BBS menu. This will help you decide whether or not you want to download the latest software.*

See Also

Info/Revisions, Info/BBS

INFO/REVISIONS

Description

See what changes or enhancements were made for a particular revision of software.

Application

This command is used for information only. It lets you know whether or not a particular bug or new feature has been addressed in the time that has elapsed since your previous software update.

Operation

Just type in the range of software revisions you wish to inspect and the results will be displayed in the main window. The date and software revision number is displayed with a list of all the changes that were incorporated in that particular version of the software.

- *When you access the BP Microsystems BBS (see* Info/BBS*), you may inspect a merged version of this list and the new chips generated by the* Info/NewChips *command. Choose the* Bulletin *command from the main BBS menu. This will help you decide whether or not you really need to download the latest software.*

See Also

Info/NewChips, Info/BBS

JOBMASTER COMMANDS

Once JobMaster is installed, the initials JM will appear on the status line near the bottom of the screen, following the programmer designation. The main menu, at the top of the screen, will include *JobMaster*. Selecting JobMaster will allow you access to the following options.

JOBMASTER/CONFIGURE

Description

Configure jobs that will be run under the *Operator mode* in JobMaster.

Application

Use *JobMaster/Configure* to set up different options to use within specific jobs created for programming.

Under *Configure* the first item is the *Job Database Directory*. This directory can be set to allow multiple systems to access the same database on a network.

Use Categories lets you decide whether or not to define a *Categories* field when you start a new job. We recommend making this decision once the first time you use JobMaster.

Default Mode lets you lock the programmer in *Operator Mode* on startup, and password-protect access to the rest of JobMaster's features. See *Locking the Programmer in Operator Mode* below.

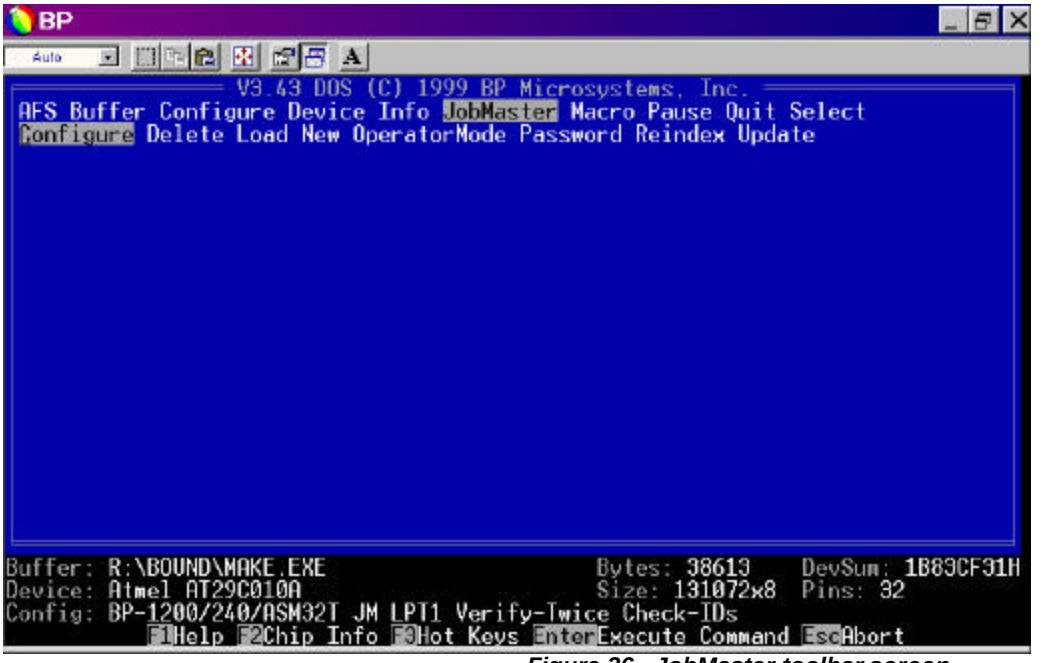


Figure 26 - JobMaster toolbar screen

JOBMASTER/DELETE

Description

JobMaster/Delete removes a job or record from the JobMaster database. It will be mandatory for you to enter the *Category* and *Item Number*, then prompts for confirmation.

Application

Use the *Delete* command in JobMaster to remove jobs or records that are no longer needed, or jobs created in error. Once a job has been removed, it will no longer be accessible.

JOBMASTER/LOAD

Description

Load performs all the functions available in *Operator Mode*, except actually programming a device. This operation is used to check, copy or modify an existing JobMaster database.

JobMaster keeps track of the date, time and author of each job's creation, and of every subsequent revision. This information is displayed whenever you select the JobMaster/Load *option.*

Application

Loading a job will prompt the BP Software to search a particular address saved while creating a job in order to load the buffer with the correct file. It will also enable the *Operator* to choose from any one of a list of parts specified for the job.

See Also

Adding a Device to an Existing Job and *Updating a Job* in *Chapter 8, Using JobMaster, page 9-1*.

JOBMASTER/NEW

Description

Selecting *New* lets you create a new job in the JobMaster database.

Application

After loading the Buffer, select *New* from the JobMaster menu and specify *Category, Item Number,* and *Devices* for a job.

See Also

Buffer/Load and *Chapter 8, Using JobMaster, page 9-1*.

JOBMASTER/OPERATOR MODE

Description

This is the mode you must select to program devices using JobMaster (see *Chapter 8, JobMaster, page 9-1*).

☐ *You can configure the programmer to start up in* Operator Mode *automatically.*

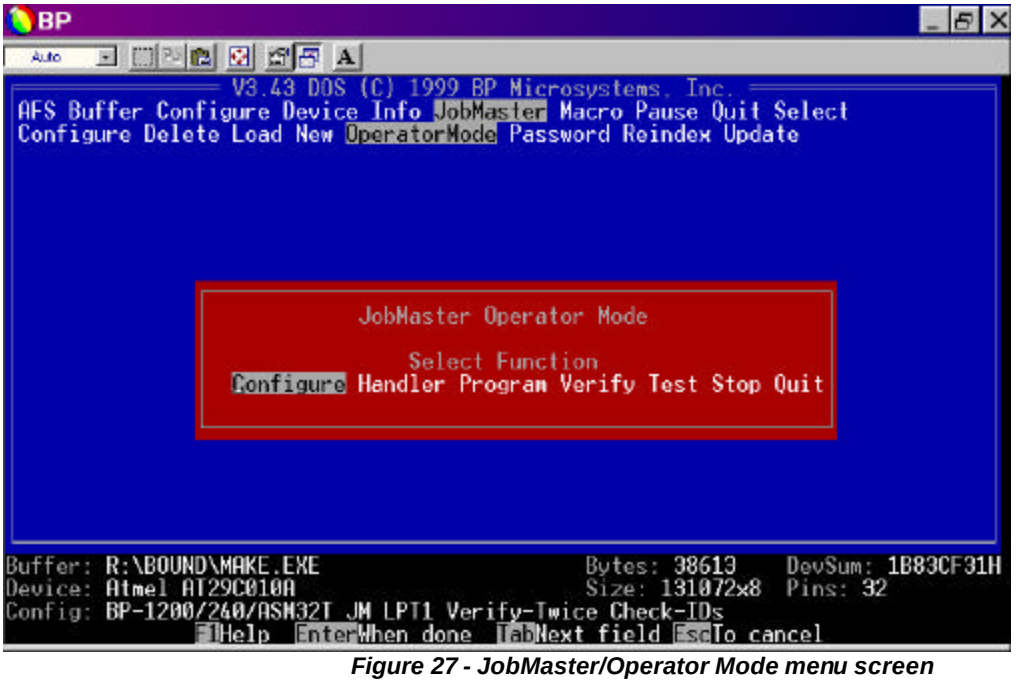


Figure 27 - JobMaster/Operator Mode menu screen

Application

When Startup in ***Operator mode*** is selected, this is the first screen the Operator will see. It covers the basic areas that the Operator will need to have access to in order to perform jobs.

The ***Configure*** option allows the Operator to configure a drive and directory and enable or disable Categories.

The Operator can change Handler types with the ***Handler*** option (if available).

Program begins the programming process after a job has been chosen.

The ***Verify*** option verifies an already-programmed device. (See *Device/Verify* in *Chapter 6*, BP *Software Command Reference*, page 6-47).

The Operator can test a particular job using the ***Test*** option.

The ***Stop*** option exits JobMaster and returns the Operator to the BP-4100/4500 main menu. If a password has been specified when the job was set up, the Operator MUST enter that password to proceed.

The ***Quit*** option exits the BP software altogether and returns the programmer's computer to DOS.

See Also

Chapter 8, Using JobMaster, page 9-1

JOBMASTER/PASSWORD

Description

Allows you to change a JobMaster password.

This function is only allowed to Administrators with the original password or when setting up a JobMaster password for the first time.

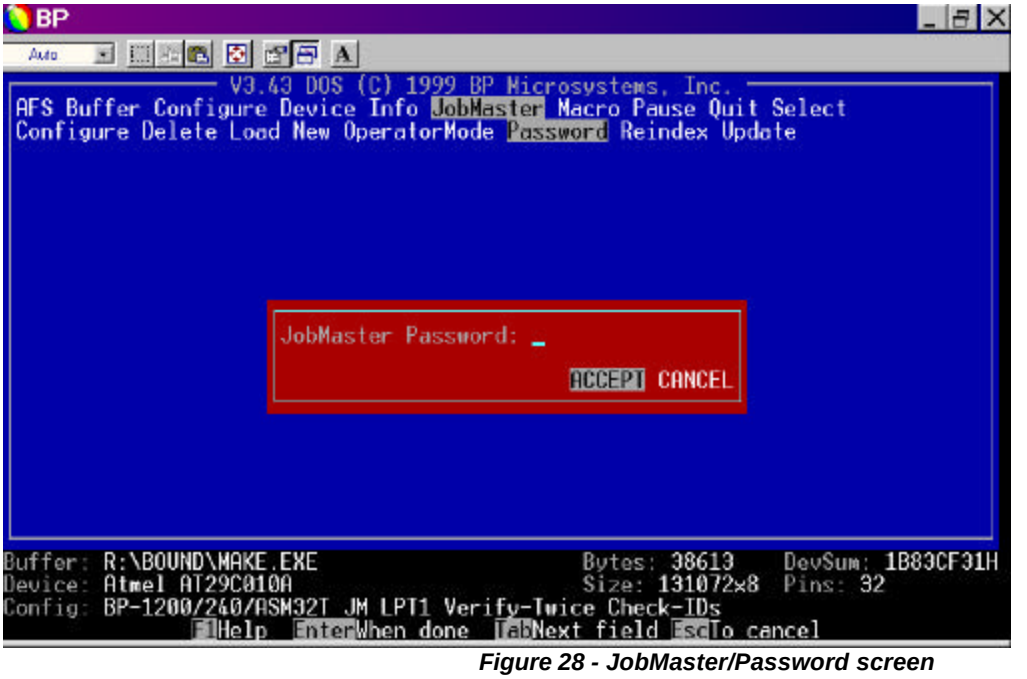


Figure 28 - JobMaster/Password screen

Application

Select *JobMaster/Password* from the menu. The BP Software will first prompt you for the current password (if presently used), then prompt you to enter a new password into the space provided. Once you have entered a new password and verified the new password, the software will store and announce that the new password has been recorded.

- **The maximum limit of characters that can be used for a JobMaster password is 15 and IS case-sensitive.**

JOBMASTER/REINDEX

Description

This option re-indexes the JobMaster database.

Application

Choose *JobMaster/Reindex* from the menu. Once <**Enter**> is pressed, the BP Software will automatically begin reindexing the jobs that have been created.

Reindex can be useful if you delete a number of JobMaster records or think the index has gotten corrupted. It will physically remove all the deleted records from the database and create a clean index.

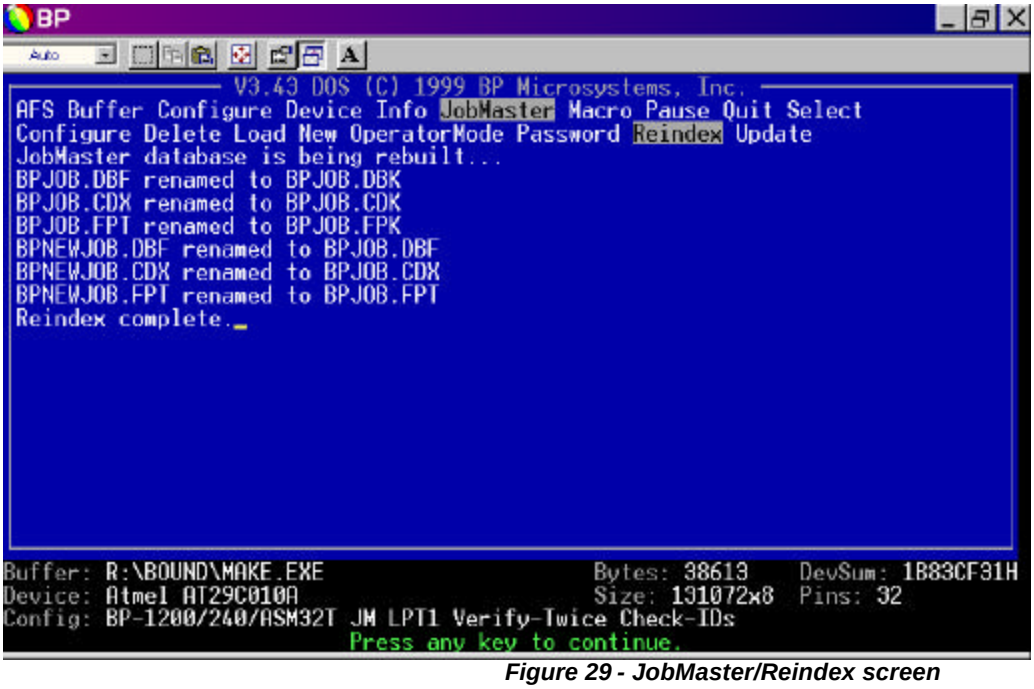


Figure 29 - JobMaster/Reindex screen

JOBMASTER/UPDATE

Description

Update saves any changes you have made to a database under the *Load* command.

MACRO COMMANDS

MACRO/DEBUG

Description

Play a sequence of commands stored in a macro file and pause after each line read from the file.

Application

This command is intended for advanced users of macro commands. It is useful when you have generated your macro files by some other means than the *Macro/Record* command.

Example: *You used an editor or wrote your own program to generate the macro files, and now you are encountering problems with the* Macro/Play.

Operation

A selector box appears showing all the macro files (.PGM) in the directory specified.

- Choose the desired directory by editing the "*Directory:*" string (*e.g.*, C:\MACROS*.PGM).
- Highlight the desired file name using the cursor up and down keys.
- Press **<Enter>** twice to execute the file.

The command sequence stored in the macro file is played back with a pause after each line read from the file and after a dialog box has been filled.

See Also

Macro/Play, Macro/Record

MACRO/FINISH

Description

End the macro record process after completing the sequence of commands needed for that macro.

This command appears only while you are recording a macro file.

Application

This command ends a macro file.

Operation

Executing the command will terminate the macro file. The last command executed prior to this command will be the last command in the macro file. When the macro file is executed, the programmer will be left in command mode once the macro file ends.

See Also

Macro/Record

MACRO/PLAY

Description

Play a sequence of commands stored in a macro file.

Hot-keys **Alt-0...9** or **Alt-F1...F9**

Application

This command saves time when doing a series of operations for a specific chip. You may record a macro file for each chip you commonly program.

Operation

A selector box appears showing all the macro files (.PGM) in the directory specified.

- Choose the desired directory by editing the "*Directory:*" string (*e.g.*, **C:\MACROS*.PGM**)
- Highlight the desired file name using the cursor up and down keys.
- Press **<Enter>** twice to execute the file.

The command sequence stored in the macro file is played back. Macro files are easily written to perform common operations, such as configuring the programmer, selecting a chip, loading a file, and programming the chip.

☐ *If you used one of the digits from **0-9** or a function key (**F1-F9**) as your macro file name, you may now use the <**Alt**> key and that same key to automatically play the macro file. This saves you time by not requiring that you execute the* Macro/Play *command every time you wish to play your macro.*

- *If you have problems with playback, the macro file may not have been recorded with the* Macro/Record *command. You may want to use the* Macro/Debug *command to determine where the error lies.*

See Also

Macro/Debug; Macro/Record

MACRO/PROMPT

Description

Displays a message to a macro file user.

- *This command appears only while you are recording a macro file.*

Application

It is used to impart information to a macro file user. You may want to create macros that tell the user what to do next, such as insert a particular chip or place a specific disk in the appropriate drive.

Operation

Enter up to four lines of text in the dialog box. When the macro file runs, the text will appear for the reader to observe. The user must press a key to continue the macro file execution.

MACRO/RECORD

Description

Store a sequence of commands in a macro file.

Application

Store commonly used command sequences in macro files to speed up operation and reduce operator errors. Macro files can be played back from the *Macro/Play* command, from the DOS command line, from a batch file, from a make file, or from another macro file.

Operation

A dialog box appears showing all the macro files (.PGM) in the directory specified:

Select the desired drive, directory, and file name to create. If you use one of the keys **0...9** or **F1...F9** as the file name, you may execute the macro by using the <**Alt**> key and the corresponding key. Otherwise, you must use the *Macro/Play* command to execute a previously recorded macro.

- You may include a comment for your own reference in the file generated by moving the cursor to the "*Purpose:*" field and entering a comment. The comment appears in the second line of the file, when viewed with a text editor.

- Pressing **<Enter>** executes the command, and the software records every command you execute thereafter.

- When finished, terminate recording by using the *Macro/Finish* command or *Quit*. If you want the macro file to leave the user at the DOS prompt after the file plays, use the *Quit* command. *Macro/Finish* lets the user perform other functions, such as chip programming.

You can include user-prompts by executing the Macro/Prompt *command while recording.*

See Also

Macro/Play; Macro/Finish; Macro/Prompt

PAUSE

Description

Execute a DOS shell.

Application

Execute DOS commands, then return to the program in its present configuration (where you left off).

Operation

The DOS command interpreter is started as a subshell in which normal DOS commands can be entered. When you want to return to the programmer, type the *EXIT* command at the DOS prompt.

If no DOS shell is created, you may not have enough available memory to generate a DOS shell. You will have to use the Quit *command instead, and then return back to the programmer by executing the .EXE file again. You may be able to free more base memory by adding EMS memory or by changing your CONFIG.SYS and AUTOEXEC.BAT files.*

QUIT

Description

Exit the program and return to DOS.

Hot-key **Alt-X**

Operation

You are returned immediately to DOS. Any buffer changes will be lost; however, if you have *"Save Configuration:"* set to AUTOMATIC in the *Configure* command, then all the options you have set during this session will be saved and restored the next time you start the software.

SELECT

Description

Specify the chip's manufacturer and part number to select the proper programming algorithm.

Hot-key **Alt-S**

Application

Selecting a device will configure the programmer for the correct programming algorithm. The algorithm is specified by the semiconductor manufacturer and includes voltage, timing, and pinout requirements. You must use this command before programming or performing any other device operations.

Operation

A dialog box appears containing a list of devices. Initially, the device list contains every part the programmer currently supports. As you type letters and numbers, the device list narrows to include only those names containing the characters you type, in the order you type them.

Example: *You can see all 27C010 parts by typing* ***27C010****. You can see all Intel 27C010 parts by typing* ***INTEL 27C010****. You can see all SEEQ parts by typing* ***SEEQ****. Pressing* ***<Enter>*** *will select the part that is highlighted. The algorithm status line at the bottom of the screen will be updated to show the part number, number of pins, and organization (number of bytes, words, or fuses if it is a PLD).*

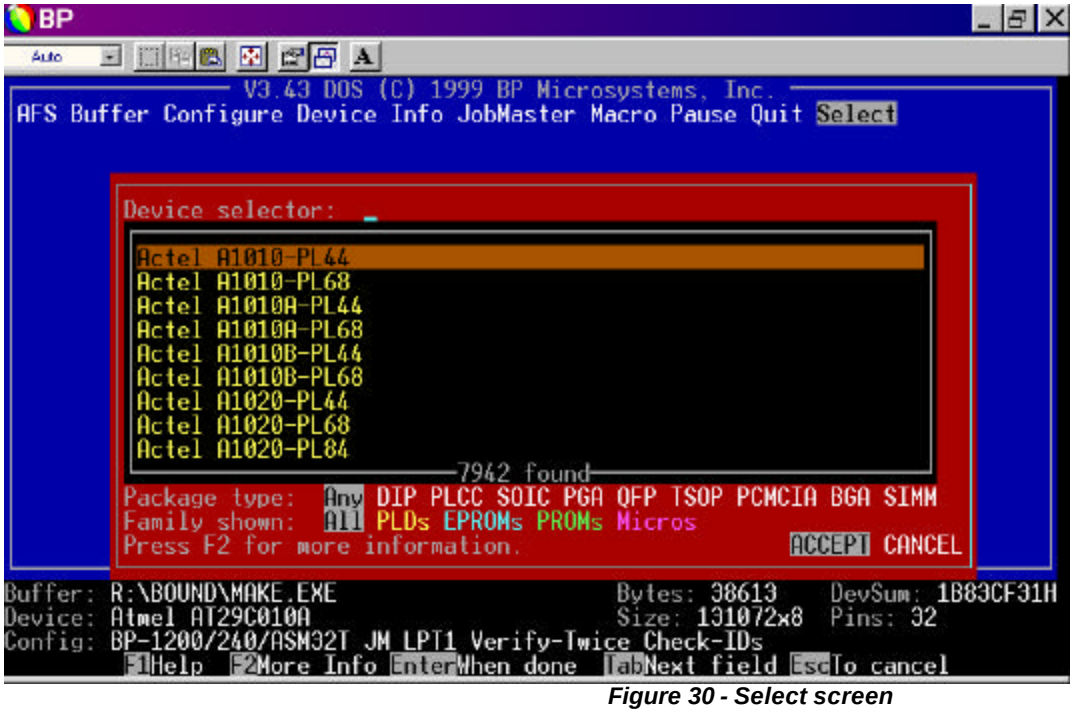


Figure 30 - Select screen

- **It is essential to choose the correct selection for the device you want to program. Programming algorithms vary widely between**

different semiconductor manufacturers and even between parts from one manufacturer with different speed ratings! Selecting the wrong algorithm can blow out your chip. The part number on your chip may have package code and temperature rating letters following the part number shown in the menu. These letters typically do not affect the algorithm you choose; however, there are some exceptions. Type the manufacturer's name and the entire part number as it appears on your chip. If no algorithm appears in the box, you may need to erase the last few letters or digits in the part number. If several algorithm selections appear, choose the one that is closest to your part's number.

Package type

You may want to use the <**Tab**> key to move to this field and select a specific package type to see what devices are supported in that particular package.

- *This is not required before programming a chip with a different package type; it is provided only for your information. It is easiest just to leave it in the "Any" position, unless you are exclusively using one package type. The programmer will automatically interrogate the socket module you have attached to determine the proper programming pinout.*

Family shown

This field allows you to narrow the device range the programmer searches for. This option lets you see PLDs, EPROMs, PROMs (bipolar), or Micros, exclusively.

- *Some device entries require special programming considerations. Pressing the* ***<F2>****, "Chip Info," key provides more detailed information about the entry, such as the name and manufacturer of any adapter that may be required to program the device. It will provide the same information displayed by the* Info/Chip *command. Once the information is displayed for the highlighted chip, you may then use the <****Up****> and <****Down****> direction keys to move through the list of chips and continue to get the information for each chip in the list.*

See Also

Device/Program, Info/Chip

CHAPTER 7
HINTS, TIPS AND OTHER USEFUL INFORMATION

ERASING EPROMS

METHOD

In order to clear data in an EPROM (in preparation for programming), the chip is exposed to short-wave ultraviolet light, which is typically provided by an EPROM eraser. The UV light penetrates a quartz window on the top of the package, erasing the data. A certain dosage is required to fully erase the part, so brighter light sources erase the part more quickly. Most erasers require between 3 and 30 minutes to erase an EPROM.

EPROMs are usually erased using a mercury vapor bulb emitting 2537 Angstroms. The bulbs most commonly used are like fluorescent tubes, but without the phosphor; they are often called germicidal bulbs. The ultraviolet light can be harmful to eyes, so erasers are equipped with shields to prevent light leakage.

PROCEDURE

Erasing an EPROM properly is not as simple as it appears. It is possible to have a partially erased chip that appears to be blank, but is not blank when read at a different voltage or temperature. Use this procedure to determine a safe erasing time:

- Start with a programmed chip.
- Erase the part in one-minute increments and use the *Device/Blank* command to test it each minute.

- Once the programmer says that a chip is blank, double the erase time to give yourself an adequate safety margin.

☐ *Some types of parts take longer to erase than others. You may need to experiment with the various parts you use. An EPROM-based part with a security bit feature (a PLD or microcontroller) is designed so that the security address will typically be the last bit to erase.*

☐ *The adhesive used on labels often blocks UV light. If the chip erases slowly, try cleaning the window with alcohol or a stronger solvent.*

☐ *Sunlight and fluorescent light can erase chips; however, this usually takes months or years. You should cover the window with an opaque label to make the data permanent.*

☐ *Some EPROM-based parts are available in inexpensive plastic packages. These parts can't be erased because they have no window. These chips are referred to as one time programmable (OTP) EPROMs.*

EMULATION MODES

16V8 AND 20V8 ARCHITECTURE

GAL stands for Generic Array Logic. It was invented by Lattice Semiconductor as a method of replacing many standard PAL architectures with a single general-purpose architecture. The GAL16V8 (20-pin DIP) and 20V8 (24-pin DIP) are EEPROM-based second generation PAL devices.

Each of the eight output pins uses an output logic macrocell (OLM) that can be configured to be combinatorial or registered, active high or active low, and can have an output enable (OE) term. This architecture is a superset of the standard PAL devices. Furthermore, the GAL's fuse map is also a superset of the standard PAL.

Programming

The programming algorithm can automatically reconfigure these two GAL devices to emulate 42 different 20- and 24-pin standard PALs. These emulation modes accept standard PAL JEDEC files.

For example, you can program, verify and test the GAL16V8 using the "GAL16V8 as 16L8" algorithm and a JEDEC file for a PAL16L8. The programmed part can be used in place of a PAL16L8.

20XV10 ARCHITECTURE

Lattice has introduced another GAL device called a GAL20XV10, which can emulate the standard PAL20L10, PAL20X8 and PAL20X4 architectures in the same manner as described above for 16V8s.

EMULATION CONSIDERATIONS

All GAL devices are programmed using generic algorithms; the programmer reads the part you put in the programmer site to determine the correct programming voltage, timing, etc. Thus, you don't need to worry about speed and power suffixes on the device you are programming.

GALs are normally guaranteed to reprogram up to 100 times, which is great for prototype design efforts. GAL devices are available from Lattice Semiconductor, National Semiconductor, SGS-Thomson and VLSI Technology.

AMD's PALCE16V8 and PALCE20V8 are 100% JEDEC compatible with the original GALs; they support all the same emulation modes.

AMD also has a 24-pin PALCE16V8HD that is not plug-in compatible with the standard 16V8, but will accept all the same JEDEC files for its emulation modes.

Changing the Socket Module with the power on

The socket module (SM) is the small metal chassis that holds the programming socket. It can be changed without turning off the programmer's power. Simply check to be sure no command is running and the Active LED is off before removing the old SM.

Testing PLCC PLDs

Some PLCC devices have a different number of pins than the equivalent DIP devices. A common example of this is the 22V10. This device is a 24-pin DIP part or a 28-pin PLCC device. The PLCC part has 4 pins that are not connected. When you generate test vectors for the device, your design software will generate either 24- or 28-pin vectors depending on whether you specified DIP or PLCC.

The programmer will automatically translate the vectors in your JEDEC file to the pinout of the part you have in the programmer site without requiring any user intervention. You simply load the file and test the chip, saving you time and potentially costly mistakes.

This also makes it easy to prototype circuits in Through-Hole Technology (THT) and to switch to Surface Mount Technology (SMT) for production.

Using Adapters to program PLCC devices

BP programmers can support most devices directly using the appropriate socket module. You may want to use a socket adapter, however, because it will allow you to program complex PLDs with a programmer that has fewer pin drivers than the size of the part.

The adapter has a DIP plug that can be plugged into the standard DIP socket of your programmer. For example, you can program an AMD MACH230 84-pin PLCC device by using the FH28E adapter on a programmer with only 48-pin drivers. The continuity test and autostart features will be disabled.

The device can be programmed and verified, but it cannot be tested. You must select the device in the device list that shows "with adapter" following the part number. This tells the software to use the DIP pinout for the adapter.

SYSTEM CONFIGURATION FOR BP.EXE

MINIMUM CONFIGURATION FOR BP.EXE

A 286 or better CPU.

At least 4MB of memory.

☐ *We do NOT require a memory manager to be present.*

If a memory manager is present, it must conform to either the VCPI or DPMI specifications. Most modern memory managers (including Windows) conform to both specifications.

CONFIGURING MEMORY MANAGERS TO RUN WITH THE DOS EXTENDER

HIMEM.SYS and EMM386.EXE (DOS 4.X Only)

EMM386.EXE must be removed from the CONFIG.SYS file or the system must be upgraded to a newer version of DOS.

Alternately, HIMEM.SYS and EMM386.EXE can be replaced with either QEMM or 386MAX. 386MAX is recommended above QEMM.

☐ *We require either the VCPI or DPMI interface in order to shift the processor into protect mode. Almost all memory managers shipped today meet the VCPI standard. Unfortunately, the memory manager shipped with DOS 4.X does not meet either of these industry standards. Upgrading to a newer version of DOS will solve the problem.*

HIMEM.SYS and EMM386.EXE (DOS 5.X Only)

Make certain that the NOEMS option **does not** exist on the EMM386 line. If it does, change the line:

```
DEVICE=C:\DOS\EMM386.EXE NOEMS [...]
```

to

```
DEVICE=C:\DOS\EMM386.EXE FRAME=NONE [...]
```

- *We require either the VCPI or DPMI interface to shift the processor into protect mode. EMM386 for DOS 5.X only supports the VCPI interface. The NOEMS option disables the VCPI interface along with the EMS interface. EMM386 for DOS 6.X does not suffer from this limitation and can use the NOEMS option without any problems.*

HIMEM.SYS and EMM386.EXE (DOS 6.X Only)

There are no known additional requirements for this configuration.

HIMEM.SYS and EMM386.EXE (MS Windows 3.1)

This is essentially the same memory manager that shipped with DOS 5.X. It should therefore be configured the same way.

Make certain that the NOEMS option **does not** exist on the EMM386 line. If it does, change the line:

```
DEVICE=C:\DOS\EMM386.EXE NOEMS [...]
```

to

```
DEVICE=C:\DOS\EMM386.EXE FRAME=NONE [...]
```

- *We require either the VCPI or DPMI interface to shift the processor into protect mode. The version of EMM386 that ships with Windows 3.1 only supports the VCPI interface. The NOEMS option disables the VCPI interface along with the EMS interface. EMM386 for DOS 6.X does not suffer from this limitation and can use the NOEMS option without any problems.*

QEMM (Version 6.X)

Verify that the NOEMS option **does not** exist on the QEMM line in the CONFIG.SYS file. If it does, change the line:

```
DEVICE=C:\QEMM.SYS NOEMS [...]
```

to

DEVICE=C:\QEMM.SYS **FRAME=NONE** [...]

- *We require either the VCPI or DPMI interface to shift the processor into protect mode. QEMM only supports the VCPI interface. The NOEMS option disables the VCPI interface along with the EMS interface.*

386MAX (Version 7.X)

386MAX supports both the VCPI interface as well as the DPMI interface. There are no known problems running with this memory manager.

SM84UP OPERATIONAL INSTRUCTIONS

The SM84UP is a universal PLCC socket capable of programming seven different pin count devices (20, 28, 32, 44, 52, 68, 84). Seven templates are included with each package pin count printed on the edge.

The socket module is plugged onto the programmer with the lid hinge toward the rear of the unit. The template edge with the number should be aligned with the front of the programmer. The front-right side of the socket module is the INDEX side. Most if not all PLCC chips have a beveled index side. This side should be aligned to the front of the socket module. Also, most will have a mark that identifies pin one. This mark is usually a small dimple or dot. When the device is inserted correctly, the dimple will be toward the front of the programmer.

32 pin PLCC devices are rectangular and thinner than the square packages. Since these rectangular chips are thinner, the plate in the lid of the socket that applies the pressure on the device must be turned over to take up the slack. Four screws with 0.059 allen heads must be removed in order to turn the plate over.

You may choose to fabricate a shim and put it on top of the device to allow for the difference in thickness of the 32 pin PLCC package type.

The SM84UP socket module can be used on a programmer with 48 pins through 240. However, a programmer with 48 pin drivers will only be able to program up to 48 pin PLCC devices. Remember, when using the SM84UP, the programmer must have at least as many drivers as the number of pins on the device being programmed. When removing the socket module from the programmer, grasp the module chassis on the left and right side. Do not remove the module by grasping the Aries socket.

GENERAL INFORMATION

The programmer will operate from 90V AC TO 260V AC, At 50 or 60 Hz. It does not need to be switched for 120/240 operation.

The programmer incorporates a special correction circuit that disables all pin drivers during power up and power down. This protects any device that may be in the programmer site when the unit is turned off, so you don't have to worry about leaving parts in the programmer site.

UPGRADING YOUR SOFTWARE

It is possible to upgrade the standard software on your programmer to the AFS software, which adds additional testing and production features, such as the autohandler interface.

To upgrade the software, first obtain an authorization code from BP Microsystems. You must now start the software and select BPMICRO as the device to program.

Use the *Device/Upgrade* command to enter the authorization code. The program will modify its own BP.EXE file to enable the option. Verify that AFS has been enabled by looking at the bottom of the screen, which will indicate AFS on the status line.

CHAPTER 8
USING MACRO FILES

WHAT IS A MACRO FILE?

A macro file stores a sequence of commands and settings. When you play back a macro file, the sequence of commands is executed. Typically, a macro file is used to select a specific chip, load a specific file, and program the chip. Macro files can be started from the menu, the DOS command line, a batch file, a make file, or a user application program. It is possible to read back a return code from the programmer that tells if the operation was successful. Macro files can be generated with an editor or automatically by using the *Macro/Record* command. Macro files are human- readable ASCII files and their name should end with ".PGM".

COMMON USES OF MACRO FILES

Production users often employ an engineer to produce a first article device while recording a macro. The saved macro file provides an automated way for a technician to reproduce the part with a minimum of time and training.

Third-party software developers can build a seamless interface to the BP Microsystems programmer to program a device when required. The software can generate a macro that will load the correct file, select the correct device, and prompt the user to place the part in the socket, all automatically.

Hardware and firmware development engineers can write a macro file to speed up the edit-compile-program-test cycle. A macro file can perform a programming sequence and return an error code to the make file or batch file that initiated the compile and program operations.

Commonly used setups can be stored in macro files and replayed with a single keystroke, saving time and errors when switching between different configurations to program different designs.

MACRO FILE CAPABILITIES

A macro file lets you execute any command on the programmer and automatically supply responses to all dialog box queries. The responses can supply a specific value, let the user supply a value, take a parameter from the command line, or leave the present value unaffected. Macro files can be nested (one file calls another and resumes execution when the second file finishes). A macro file can prompt the user with a message that you supply. Finally, you can have the programmer software automatically generate the macro file; you can modify one that was generated; or you can write one from scratch (using an ASCII editor or writing it directly from your application program).

INVOKING MACRO FILES

There are four ways to invoke a macro file: 1) from the DOS command line, 2) from the *Macro/Play* or *Macro/Debug* command, 3) from a hot-key, or 4) from another macro file.

BATCH MODE, MAKE FILES

A macro file can be started from the command line, so it can be included in batch files, make files, and user programs. To run a macro file from the command line, just specify the macro file name on the command line. It is not necessary to specify the .PGM extension. See below for example (where "bp" is the name of your programmer's executable file and "example.pgm" is the name of the macro file):

bp example

If your macro file uses command line parameters, you can specify them here as follows:

bp example parm1 parm2 parm3

START FROM COMMAND MODE

You may start a macro file using the *Macro/Play* command. In this case, choose the file you want to execute by moving the highlight over the file name and pressing **Enter** twice. The *Macro/Debug* command can be used for debugging purposes.

HOT-KEYS

Instead of using the *Macro/Play* command every time you wish to execute a macro from the command mode, you can use the **Alt** key in combination with a single digit or a function key. However, the macro must be named as

that digit or function key followed by the standard .PGM extension. You can use the 0-9 or the F1-F9 keys.

FROM ANOTHER MACRO FILE

While recording a macro you may use the *Macro/Play* command to call another macro. This will simply record the act of executing another macro and will not record the steps carried out by the nested macro. When played back, the macro interpreter will simply execute the *Macro/Play* command just as any other command and play the nested macro until it is finished and then return to the original macro and continue. You may nest as many macros as you like, with the only limitation being the number of FILES (set in your CONFIG.SYS) you may have open at one time.

GENERATING MACRO FILES

The easiest way to generate a macro file is to execute the *Macro/Record* command which will allow you to record commands and dialog box responses in a macro file. When the file is replayed, it will execute those commands without any superfluous user input. The file can be edited with an ASCII editor to make slight changes such as incorporating command line parameters or allowing user input.

During playback, the macro file provides responses for command selections and dialog boxes. Input is taken from the user any time there is an error or a new chip is required. Hence the user is required to respond to prompts such as "Abort, Retry, or Ignore" and "Insert Chip."

QUIT OR FINISH

The *Macro/Record* command is terminated by selecting either the *Quit* or the *Macro/Finish* command. Selecting *Quit* causes the programmer to return to DOS upon completion; useful when using batch file or make file commands to control the programmer. The *Macro/Finish* command (appears only while recording) is useful when you want to retain manual control of the programmer (in order to program chips) after executing a sequence of commands (e.g., *Select, Device/Options, or Buffer/Load*).

NESTING

Macro files may be nested. While using the *Macro/Record* command, you may play a macro using the *Macro/Play* command. The selected macro will play to completion, then allow you to continue recording the new macro file. If the macro you have selected to play back ends with the *Quit* command, the *Quit* will be ignored so you can continue.

PROMPTS

The *Macro/Prompt* command can also run while recording a macro. It allows you to enter up to four lines of text that will be used to prompt the macro file user. When the macro file is played, the text will appear on the screen and the user will have to press a key to continue.

OTHER CAPABILITIES

Creating a macro file that accesses extended capabilities such as command-line parameters, and user responses requires the use of an ASCII editor to modify files created with *Macro/Record*. Study the example file and the file format specifications below to understand these features.

MACRO FILE FORMAT

Macro files use a simple ASCII format. There are four record types defined: 1) the header, 2) the command record, 3) the data record, and 4) the comment. You can edit the file with a standard text editor or a word processor in non-document mode.

EXAMPLE MACRO FILE

An example file will illustrate the file format and the use of command line parameters. This example will select the INTEL 27256, load a file named TST32KB.BIN, set device options that require user response, program the chip, and return to DOS. The file is named EXAMPLE.PGM:

```
;Macro file V2.20 for BP-1200 generated 02/09/93 17:41:11.
;
/Select
    Device selector: %1
    Package type: Any
    Family shown: All
    : ACCEPT
/Device/Options
    Starting word of range: !0x100
    Ending word of range: !0x200
```

Buffer offset: ?0x0

Clear buffer before reading: YES

Data path width (number of EPROMs): 0x1

Number of banks: 0x1

Programming mode: SET

Blank check before programming: ENABLE

Verify after programming: TWICE

Continuity test: ENABLE

Check electronic identifiers: ^ENABLE

: HEX

: ACCEPT

/Buffer/Load

Directory: C:\JED\TST.**

File to load: %2

Type: BINARY

Clear buffer before loading: YES

Lowest address to load: 0x0

Highest address to load: 0x3fffff

Load address in buffer: 0x0

: HEX

: ACCEPT

/Device/Program

/Quit

: ACCEPT

;End of file

To run the macro file from the DOS command line, you just type:

bp example "intel 27256" "tst32kb.bin"

If there are no errors, the user doesn't have to press any keys. If an error occurs, the user must press a key to continue. The program will return an error code to DOS that can be examined by a batch file or a user's application program. If no error occurs, the program will return 0. See Chapter 9, *Trouble Shooting*, for list of error codes and their meaning.

MACRO FILE HEADER

The macro file must start with the following text:

Macro file V...

This unique string lets the programmer identify the file as a valid macro file.

COMMAND RECORD

The command record is a line with a slash (/) in column one, followed by the command name. Commands selected from submenus have the main menu entry first, a slash, and the command name last. Example:

/Buffer/Load

Any command may be invoked by using a command record. Editor commands (fill, copy, checksum, etc.) are represented by command records even when the user invokes the commands with function keys (see *Editor commands)* below.

DATA RECORD

The data record is a line starting with a space or tab character, also containing a colon (:) character. Following the initial white space is the *prompt* field. It extends up to the colon. To the right of the colon is the *response* field. The response field may contain an ASCII string, a button (the highlighted text of a multiple-choice line), or a number. Strings may be empty. Buttons must match the exact text shown in the dialog box. Numbers may be either hex or decimal: hex numbers have the 0x prefix; decimal numbers do not.

COMMENTS

Comments may be placed on any line in the file. Comments begin with a semicolon (;) and continue to the end of the line. Also, blank lines may appear at any point in the file.

DIALOG BOXES

Each dialog box entry is represented by a data record. The data records must appear in the macro file following the command record that invoked the dialog box. The data records must appear in the same order as the entries in the dialog box. The prompt field must be identical to the prompt appearing in the dialog box. If there is a response in the dialog box without a prompt (e.g., ACCEPT/CANCEL), then the colon appears before the response without a prompt (e.g., :ACCEPT). For these reasons, it is easiest to generate a macro file using the *Macro/Record* facility to be ensure that the prompts match exactly.

COMMAND LINE PARAMETERS: %1, %2,...

A command line parameter may be substituted at any point in the macro file. It is only possible to use command line responses when the macro was started on the command line:

bp macrofile foo bar c:\hex\test.hex

Use a percent sign (%) and a digit (0-9) to specify a parameter. Here is an example of a data record:

Directory: %3

Will be translated into:

Directory: c:\hex\test.hex

This is useful when you want to specify a chip name or a file name from the command line as shown in the EXAMPLE.PGM file above.

OPTIONAL DATA RECORDS: '!'

The '!' is useful for data records that may or may not be present when the dialog box is displayed by the macro. It allows a field and its response to be used if the field is present for the particular chip and will not generate a warning for a chip that does not possess this field. In EXAMPLE.PGM above, this was used on the *"Starting word of range:"* and the *"Ending word of range:"* fields because they are present for most EPROMs; however, some Flash devices require complete erasure and do not offer this feature. Suppose the exclamation (!) characters were not embedded into EXAMPLE.PGM and the following was executed at the DOS prompt:

bp example "Intel 28F010" "tst32kb.bin"

The user would receive a warning each time a range field was encountered in the macro file, because those fields are not present when the "Intel 28F010" is selected. The '!' allows the above example to run without interruption.

USER INPUT: '?'

To make the macro stop and request a response from the user, place a '?' in front of the first character of the response portion of the field. This will leave the response in the macro file as the default value, possibly allowing the user to simply confirm by hitting **Enter**. The '?' may replace the response completely and will require to user to type in a new result.

In the EXAMPLE.PGM above, a '?' was used on the *"Buffer offset:"* field to allow the user to specify the buffer offset depending on which chip he/she is programming.

LEAVING A FIELD UNTOUCHED: '^'

Specifying a caret (^) character as a response will leave the field unmodified. Deleting the field from the macro file will achieve the same result; however, it may not be as obvious to someone who updates and maintains the macros. This allows the user to set certain options that he/she can be assured want get changed by running a macro.

We placed a '^' on the *"Check electronic identifiers:"* field because it would be annoying if the user deliberately disabled this feature for a chip with a bad ID; and every time he/she ran this macro, it set it back to enabled, checked the ID, and halted the macro. Here we are assuming the user has better judgment over this option than a macro recorded under perfect conditions.

EDITOR COMMANDS

The buffer editor may be invoked from the macro file. Once in the editor, command names represent the keystrokes that execute editor commands (checksum, fill, copy, etc.). Also, data on the screen can be modified with an additional command: modify. The following commands are available in the editor:

/set_address

/reconfigure

/search

/fill

/copy

/invert

/checksum

/return

/modify

The return command is used to leave the editor (equivalent to pressing **Enter, Esc,** or **F10**). The modify command is used to modify data in the buffer (equivalent to typing over data fields). The modify command requires a single data record as follows:

byte: addr bits position data

Where the four fields are hex numbers (do not precede with 0x) and have the following meanings:

addr	*the byte address to modify*
bits	*the number of bits that will be modified*
position	*the bit position to modify (location within the byte where bit 0 of data will be placed)*
data	*bit pattern to place into the byte.*

Example: buffer byte 3BC (hex) contains 7f and the following commands are executed.

/modify

byte: 3bc 4 3 2

Has the following effect:

3BC: 7f 01111111 (before)

3BC: 17 00010111 (after)

You can use the *Macro/Record* facility to generate these editor commands and then modify the resulting macro file to suit your needs.

CHAPTER 9
USING JOBMASTER

OVERVIEW

JobMaster is a powerful tool for automating production. It allows an Administrator to set up a job to precise specifications, test the results and then protect the routine so that it cannot be inadvertently modified. Once a master device is programmed and approved, JobMaster is ready to begin full-scale programming immediately, without further set up.

JobMaster relates to two different types of people within the production facility:

- Administrator(s) – designated to set up jobs within the software, as well as regulate changes made to any existing jobs. Under *Operator Mode*, these functions can only be done by an Administrator possessing the password.

- Operator(s) – runs the jobs to program devices.

FEATURES

JobMaster has been designed for maximum simplicity and consistency. It takes the "guesswork" out of many of the steps needed to program devices, such as choosing a device, selecting buffer contents, setting special configuration needs, etc. Once a job has been set up, the Operator's only tasks are to specify the number of devices to program, and press <**Enter**>. Configuration, loading the buffer and device selection are taken care of automatically.

JobMaster sufficiently takes the place of using macros. Originally, macros were the standard for creating a recorded file of commands and settings to run a programming function. The JobMaster software, supplied with an automated system, incorporates the usability of the macro function with enhanced features and supplies a way to monitor and secure all jobs created and stored within the software.

Since JobMaster can be password protected, it also ensures that Operators can only run predefined jobs, which further cuts down on the possibility of Operator errors. When *Operator Mode* is selected for startup, a window will appear showing only those functions allowed without a password.

JobMaster also supplies a user-definable *Note* field for additional information that the Operator may need to know before performing a job. When a job is chosen, a window will appear displaying any information contained in this field.

JobMaster also helps to maintain a record of jobs done by providing a *Category* and an *Item/Part Number* field for each job created. So, essentially, a job can be created and stored, easily found and tracked by its *Category* and *Item/Part Number* information.

Example: *Company X, your customer, wants to program 500,000 devices with a file they have sent you, named Prog1. You, the Administrator, select the device to be programmed and load the buffer from the floppy drive, then list the new job under the Category, Company X, and Item/Part Number, Prog1. After the job has been completely set up, tested and secured, you assign an Operator to perform the task. Now, the Operator need only to load the job found under the appropriate Category and Item/Part Number and begin programming devices.*

Say Company X calls back and orders a different job done. For tracking purposes, all you, the Administrator, have to do is place this order under the same Category as the first with a different Item/Part Number. Now when the Load option is chosen from the JobMaster submenu, two Item/Part Numbers will appear in the Company X Category list.

With the above listed example, it is easy to see how JobMaster assists in tracking jobs and storing client information.

JobMaster also tracks the original author of each job and all changes to jobs. This information pops up in a window after the *Note* appears when a job is loaded. This information is helpful in checking the status of jobs that have been edited or updated as well as keeping track of when revisions are made to jobs and by whom.

JobMaster supports multiple substitution of devices for each job, allowing for one original *Device* field (the device for this field is chosen prior to the creation of a new job) and five alternates for a total of six *Device* fields within any one particular job.

Since the buffer is loaded prior to creating a new job, JobMaster tracks the path to locate this file, enabling files for buffer loads to be stored on network drives, local PCs or floppy disk. Once a job is loaded, JobMaster will go in search of the file with the provided path way saved previously and will prompt you should the path name be invalid or incorrect. JobMaster's databases can also be shared company-wide, allowing for the ease of file sharing.

JobMaster allows you to create a job on a programmer, then run it on an autohandler if needed. The only restriction is that laser-marking information can only be added to the job if the laser marker is attached to the programmer. However, the job can be created on another programmer and the laser information added (in a relatively short amount of time) once the job has been transferred to a programmer utilizing an autohandler.

INSTALLATION

JobMaster is supplied within the BP Software; however, you must have an access code to obtain usability. To receive an access code, you must purchase the JobMaster software from our sales department (contact information listed in *Chapter 10, Troubleshooting and Maintenance, page 14-1*). Once you have purchased the software, an access code will be generated and either e-mailed or faxed to you.

After obtaining access to the software, you need to:

- Configure Jobmaster - this is done through the *JobMaster/Configure* option, listed on page 6-55.
- Set the LIBPATH environment variable to include the directory where BP.EXE and BPJOB.DLL reside.
- Set the BPCFG environment variable to your local hard drive.

  ✓ **Do *not* share the BP.CFG files among multiple PCs.**

- Reboot your PC and run BP.EXE
- Select the Upgrade option from the AFS menu and enter the upgrade code that was shipped with your diskettes or provided by the BP Microsystems Sales Department.
- Go to the JobMaster/Configure menu and set your JobMaster Database Directory. This will be the directory where all of your users can access the database. Choose whether or not you want to enable categories and tab to Accept.
- Press <**Enter**> and your options will be set.

CREATING A NEW JOB

The procedure for adding a new job to the JobMaster database is very straightforward. This section assumes you understand the basics of programming a device using the BP software. The steps for programming a device are detailed in *Chapter 6, BP Software Command Reference*, using different sections pertaining to input/output devices. Individual commands and functions are listed in Chapter 7 – Reference Commands. Before creating a new job within JobMaster, make sure you have taken the following steps:

- Select the device that you want to program.

- Load the buffer with the information to be programmed into the device.

- Set any special configuration options.

Once you have chosen at least one device and have loaded the buffer contents and set any special configuration, you are ready to create a new job. Go to the *JobMaster* menu and select the *New* option.

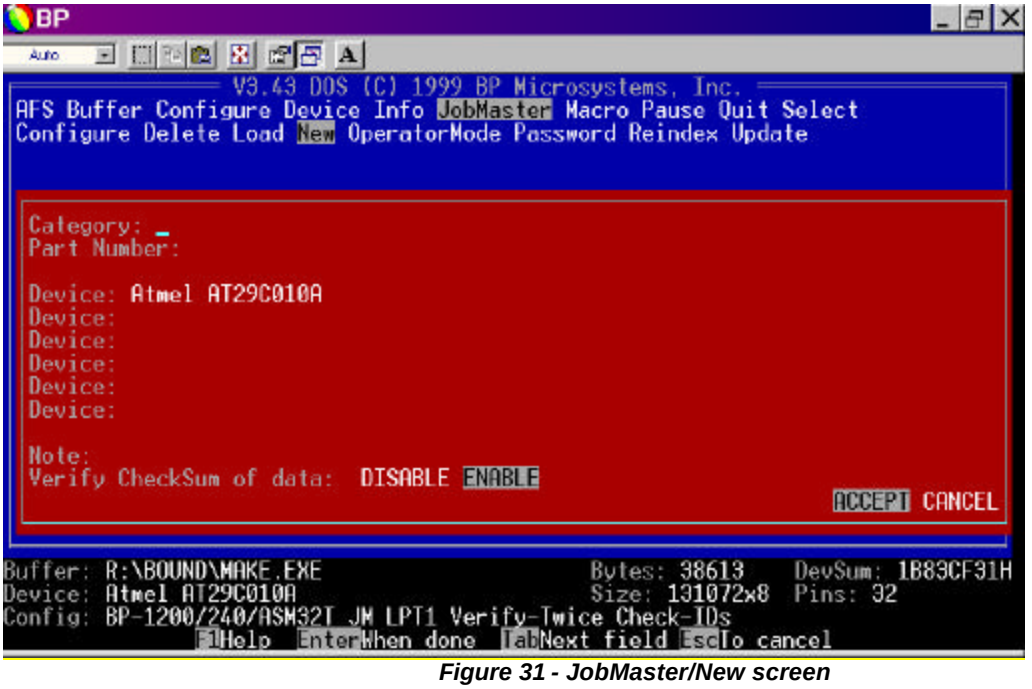


Figure 31 - JobMaster/New screen

You can define the *Category* and *Item/Part Number* fields in any way you choose.

We suggest the following options for filling in these fields:

- *Referencing a manufacturer in the* Category *field, if you will be creating specific manufacturer databases, such as Category: AMD to be saved in the AMD database*

- *Referencing your customer in the* Category *field and the job number or location on the circuit board in the* Item Number *field.*

When the Operator chooses *Load* from the JobMaster option bar, a window listing all categories will appear for the Operator to choose from. Once a category is chosen, a window will appear listing all item numbers associated with that category. So, by creating *Category* and *Item Numbers* that will be familiar to the Operators, you help to ensure the correct job is chosen. The

Category and *Item Number* will also be displayed whenever the Operator selects a job, thus providing another means to double-check that the correct selection has been made.

The device you have programmed is automatically entered into the first *Device* line. You can add up to five additional devices, provided they have identical programming requirements. See *Adding a Device to an Existing Job* below.

The *Note* option is another user-definable field that will be displayed whenever an Operator chooses this job. You may use this field to inform the operator of any information that might be useful to the job, or you may leave it blank. If the field contains information, that information will pop up on the screen for the Operator when a job has been selected.

If the *Verify Checksum of Data* option is **Enabled**, the system will stop and not permit the job to run if the data has changed since the job was created.

- **We strongly recommend that this option be set to *Enabled*, unless you have a data file that is going to be changed frequently.**

Once you have set the options, select ***Accept*** and press **<Enter>**. A screen will prompt you to verify that you want to add the new job to the database. If the information is correct, press **<Enter>** again to save it in a database. If changes need to be made, selecting *Cancel* will take you back to the New Job screen.

COPYING AN EXISTING JOB

To copy the *Category*, *Item Number* or *Device* from a currently existing job, press **<Enter>** when the cursor is on an empty field. This will bring up a selection box that you can use to select currently existing entries. Highlighting a selection and pressing **<Enter>** will input the information into the new job.

ADDING A DEVICE TO AN EXISTING JOB

JobMaster permits the programmer to list as many as six devices under a single job description. This can be a real convenience in a production environment, allowing the use of devices from multiple manufacturers.

- **It is essential that all the devices listed for a single job be absolutely compatible. They must accept exactly the same configurations and programming algorithms.**

Example: *You cannot mix a PLD with a Flash EPROM.*

To add a device, select *JobMaster/Load* to load the job. Then, using *JobMaster/Update,* bring up the change screen and add the device to the next available device line by tabbing to an empty device line and pressing **<Enter>**. This will bring up the device selector dialog screen. Once all of your changes are complete, select *Accept* and press **<Enter>**. The updated job will be saved over the existing job in the database.

UPDATING A JOB

Go to *JobMaster/Load* to select and load a job. Use the BP software to change the *Device/Configuration, Buffer/Options* or other parameters, then select *JobMaster/Update* to accept the changes as part of the current job.

JobMaster keeps track of the date, time and author of each job's creation, and of every subsequent revision. This information is displayed whenever you select the JobMaster/Load *option.*

LOCKING THE PROGRAMMER IN OPERATOR MODE

One of the major highlights of JobMaster is the ability to secure the programmer and limit the Operator's access to certain functions through the use of *Operator Mode.* To lock the programmer in *Operator Mode*, select *JobMaster/Options* and set the default mode to *Operator Mode*. Tab to *Accept* and press <**Enter**>.

When you select the Operator Mode *default option, you will be prompted to enter a password. If you enter a password, it will be necessary to use it to get out of Operator Mode (see* Running a Job *below). If this protection is not desired, do* not *enter a password, tab to* Accept *and press <**Enter**>.*

PASSWORD PROTECTION

In a typical production environment, it can be useful to limit password access to those people who are authorized or qualified to make changes to the job specifications.

The *Operator Mode* selection is stored both in the programmer itself and in the BP software. As a result, even if a different PC is attached, that programmer will still be locked in Operator mode. Similarly, the software on any PC that is attached to a programmer locked in *Operator Mode* will become locked also. In either case, the password will be necessary to unlock that programmer and exit the Operator Mode.

This feature is very useful in preventing inadvertent or unauthorized alteration of the programming algorithm and database.

- **Please make sure to write your password down and put it in a safe place. If your password is forgotten, a field-service technician will need to come out to reset the password entry for you.**

RUNNING A JOB (PROGRAM MODE)

JobMaster is designed to make programming as efficient and foolproof as possible.

Follow the instructions in the preceding chapters for getting started and position the correct input and output media on the programmer.

Assuming the *Operator Mode* default has been selected, the programmer will "wake up" in *Operator Mode.* If not, refer to the *Locking the Programmer in Operator Mode* section above. *Operator mode* offers only four options: *Program*, *Verify*, *Stop* and *Quit*. To begin programming, tab to the Program option and press **<Enter>**.

1. Tab to *Program* and press **<Enter>**.

2. JobMaster will ask for a *Category*, *Item Number* and *Device* to program. Enter these at the prompts. You can always press **<Enter>** on an empty field to see a list of available options from the database. If more than one device has been specified for the job, you will be prompted with a list from which to choose. Simply tab to the appropriate selection and press **<Enter>**.

3. Once you have selected the device, JobMaster will display any Notes that were added when the job was created.

4. The software will then ask for the *Number of Devices*. Type this number and press <**Enter**>.

   This refers to the number of devices that should be successfully programmed, **not** *the number of attempts, to allow for any rejects that may normally occur*.

5. A summary screen will be displayed, listing *Category*, *Item Number, File, Checksum, Operation* and the *Note.* If everything is correct, select *Accept* and press **<Enter>**. The system should begin programming devices.

DELETING A JOB

To delete a job from the JobMaster database, you must have access to the Administrator password.

1. Go to JobMaster/Delete.

2. Select the Category and Item Number of the job you wish to delete.

3. A screen will appear asking you to verify the delete option.

4. Press <**Accept**> to continue with the delete process; choose <**Cancel**> to cancel the delete process.

RETURNING TO NORMAL MODE

To return to the BP Software to normal mode upon startup, simply enter the Administrator password, and then go to *JobMaster Configure*. Tab down to the *Default Mode* field and select <**Normal**>. When the BP Software is started up again, it will be in normal access mode.

For a complete list of JobMaster commands and descriptions, refer to *Chapter 6, BP Software Command Reference*, page 6-1.

CHAPTER 10
SERIALIZATION

This chapter describes the basic procedures for using the optional Serialization feature available for the BP Microsystems' programmers. This feature supports simple serialization: a linear sequence of numbers incremented by one. It also supports complex serialization for users who wish to generate a unique sequence of numbers, including non-linear sequences, by writing their own algorithm.

To use the Serialization feature, you must have BP software V3.27 or newer and purchase the Serialization option.

- **Although this feature works with 98% of all EPROMs and Microcontrollers, it DOES NOT work with PLDs.**

For information on obtaining this option and updating the software and hardware, contact BP Microsystems' sales department, contact information provided in *Chapter 10, Troubleshooting and Maintenance*, page 14-1.

OPERATION

1. From the BP main menu, select *AFS* and press <**Enter**>.

2. From the submenu, select *Serialize* and press <**Enter**>.

3. On the screen you will be given three serialization options: **NONE**, **Simple**, and **Complex**. Use directional arrow keys on your keyboard to select the option you prefer and press <**Enter**> to accept.

4. Depending on the option you selected, you will be prompted to provide certain essential information.

NONE

Choosing this option disables serialization completely and requires no additional information.

Simple

This option instructs the BP software to generate a linear sequence of numbers and requires that you input the following information:

- *Serial number size*:
  Select from *NONE, 1_byte*, *2_bytes* or *4_bytes* and tab to the next selection.

- *Serial number data file:*
  Enter the path and name for the data file that will contain the current serial number. The BP software automatically updates this file as each new number is generated. This permits the user to interrupt a job and continue it later with the next number in the series. Now tab to the next selection.

  ✍ **The *Serial number data file field* should identify the ".dat" file used by the BP software to keep track of numbers for simple serialization. DO NOT enter the name of the ".exe" file used to create numbers in the complex serialization mode! This will cause that file to be over-written and destroyed.**

- *Buffer address for serial number:*
  Enter the address at which you wish to place the serial number. The default address is 0 (zero). Now tab to the next selection.

- *Buffer bytes to skip*:
  Select from two options, *NONE* and *EVERY_OTHER*. This is useful, for example, when programming paired 8-bit EPROMs that will represent one 16-bit address. The EVERY_OTHER option allows all of the serial number to be written into one of the two chips in the 16-bit set. Tab to the next selection.

- *Byte order:*
  Select from the options **REVERSE** and **FORWARD**. **Reverse** means the least significant byte (LSB) comes first. **Forward** means the most significant byte (MSB) comes first. Tab to the next selection.

- *DECIMAL HEX:*
  This option specifies whether the *Buffer address for serial number:* entered above is to be interpreted as decimal or hex. Select the correct option and tab to the next selection.

- *ACCEPT CANCEL*
  If you are satisfied with your choices, select *ACCEPT* and press <**Enter**>. If you have specified a serial number file that already exists, skip to "Working with an existing file" below.

  If a new serial number file is being created, this will bring up a screen, which shows *Serial number file created, starting at Serial Number 1.*

  By pressing any key, you will bring up another screen that says *Starting Serial Number:* ***1,*** which is the default starting number for a new file.

  1 is the default starting number. Press <**Enter**> to accept this, or change the starting number and then press <**Enter**>. Now you can press <**Esc**> which will take you back to the main menu, and begin programming.

Working with an existing serial file

If you have specified a serial number file that already exists, neither of the previous screens will appear, and the starting serial number will be whatever was stored in that file. Tab to the next selection.

- *DECIMAL HEX*
  Select the format in which the serial number is to be represented then tab to the next selection.

- *ACCEPT CANCEL*
  If you are satisfied with the information you have input, select *ACCEPT*.

Now you can press <**Esc**> to return to the main menu and begin programming.

Complex

If you choose the *Complex* option, you must first have created a program to generate the unique sequence of numbers desired. This can be a linear sequence such as "increment by five" or a completely non-linear sequence. Details of the protocols involved will be covered in a following section. Assuming this program already exists, here is the procedure for using Complex Serialization:

The screen will prompt you for the following information:

- *First Serial Number:*
  You must specify the beginning number in the sequence of serial numbers. This information is passed to your Serialization program.

- *Last Serial Number:*
  This field is optional. Any number you enter will represent the maximum value desired. If a value is entered in this field, then it is passed as a

command-line parameter to the user's serial-number generating program. It is the burden of that program to output the appropriate error condition to the file it generates. Leaving this field blank will disable this feature.

- *Serialization program:*
  This field must contain the full path and name of the user's Serialization program.

- *ACCEPT CANCEL*
  If you are satisfied with your choices, select ***ACCEPT*** and press <**Enter**>.

Now you can press <**Esc**> to return to the main menu and begin programming.

ALGORITHM PROTOCOLS

To use the Complex Serialization option, you must first create a program that will generate the unique sequence of serial numbers. The path and name of this program must be specified in the set-up procedure as outlined above. The BP software supplies this program with the Command-line Parameters outlined below, and the program supplies the BP software with data in the form Txx, where "xx" is the appropriate code number, followed by the data in ASCII representation.

See example serialization program at the end of this chapter.

COMMAND-LINE PARAMETERS

The BP Software supplies the user's program with three command-line parameters.

- **-N <Serial Number>**
  This is the current serial number, in ASCII representation.

- **-E <Serial Number>**
  This is the ending serial number, in ASCII representation. This parameter will only be passed to the user's program if the *Last serial number:* field has been filled in. If so, the user's program must specify an Error Message to be generated (see below under "Optional Codes - T06").

- **-F**
  This command-line parameter tells the program that this is the first time it's being called. It is only passed to the user's program when it is being called for the first time in a given session. Subsequent calls to the user's program will not pass this parameter.

File Codes

The Serialization program must create a file to pass information to the BP software. This file must be named SERIAL.DAT.

The Serialization program should not specify any path information. This will cause the file to be created in the active directory, where the BP software will look for it. The active directory is not necessarily the same directory as the BP software.

The SERIAL.DAT file elements each begin with a code. These are in the form of "Txx:" where "xx" represents the appropriate code, followed (with no space) by the data in ASCII representation. These elements inform the BP software to perform specific tasks.

Four of these elements are essential, the rest are optional.

Essential Codes

- **T01** Current Serial Number
- **T02** Next Serial Number
- **T03** Translation format number, specifies the file format (See the following chart for supported formats and their codes.)

SUPPORTED FILE TYPES For Serialization on BP-4100/4500	
File Types:	**T03-Codes**
Absolute Binary	16
ASCII Hex Apostrophe	52
ASCII Hex Comma	53, 58
ASCII Hex Percent	51, 56
ASCII Hex SMS	57
ASCII Hex Space	50, 55
Fairchild Fairbug	80
Formatted Binary	10
Intel Hex-32	99
Intel Intellec 8/MDS	83
Intel MCS-86 Hex Object	88
Intel OMF 286	98
Intel OMF 386	97
JEDEC (full)	91

JEDEC (Kernel)	92
Motorola 32 bit (S3 record)	95
Motorola Exorcisor	82
Motorola Exormax	87
POF	14
Tektronix Hexadecimal	86
Tektronix Hexadecimal Extended	94
Texas Instruments SDSMAC	90
Texas Instruments SDSMAC (320)	04

Table 11 – Supported File Types

- **T04** Specifies the beginning of the actual data which will be outputted to the buffer. The data must be in the format specified by the T03 code. The data will continue from this point until the end of the file. If the data is in a non-addressed format, it will be put at address zero. The user may specify a different beginning address.

Example: *Here is an example of a file that the user's program will output as SERIAL.DAT, which the BP software will use to manage the serialization process.*

Current Serial Number:	***T01:D60043C8***
Next Serial Number:	***T02:D60043C9***
Translation Format:	***T03:99***
Data to Output to Buffer:	***T04:***
	:020000020000FC
	:020000040000FA
	:0600200000A0D60043C859

OPTIONAL CODES:

- **T05** Message, Fatal error, Abort.
  On receiving this code, the BP software will display a message and abort the serialization process. Programming will not continue. The content of the message, in ASCII representation, and the conditions under which this code will be issued must be specified by the user's serialization program.

- **T06** Current Serial Number greater than limit, Message, Abort.
  On receiving this code, the BP software will display a message and abort the serialization process. Programming will not continue. The content of the message must be specified in ASCII representation by the user's serialization program. This code should only be issued if the -E command line parameter has been given a value in the serialization set-up *and* that value is exceeded.

- **T11** Warning Message string.
  On receiving this code, the BP software will display a message and will

continue the serialization process. The message will be specified by the user after the T11 code, in ASCII representation. The conditions under which this message should be displayed are to be determined by the user.

- **The serialization program specifies an address range to which the serial number will be written. This must be the same range each time the program is run for a given set. If the program attempts to write to a different address range, an error message will be generated and serialization will be aborted.**

EXAMPLE PROGRAM

The following is a sample serialization number program to be called by the BP software.

```
/************************************************
*********

Example serialization program.
This is a sample serial number generating program
to be called by the BP software for serializing
devices.
************************************************
*********

#include <stdlib.h>
#include <stdio.h>

// Function prototypes
int ParseCommandLine (File *fp, int argc, char
                      **argv, long *plSerialNum,
                      long *plAddress, long
                      *plIncr);
void WriteDataInFormat (FILE *fp, long lAddress,
                        unsigned char *szData, int
                        nDataLen);

//Macro definitions
#define SERIALFILE                    “serial.dat”
#define DEFAULT_ADDRESS    (0)
#define DEFAULT_INCR                  (1)

int main (int argc, char **argv)
{
     int rtn;
     long lAddress, lIncr, lSerialNum,
                       lNextSerialNum;
     FILE *fpSerial=NULL;

     //Create the serial.dat file
     fpSerial – fopen (SERIALFILE, “w”);
     if (fpSerial == NULL)
     {
          printf (“Unable to create %s file.\n”,
                       SERIALFILE);
          return 1;
     }
```

```
      //Parse the command line arguments
      rtn = ParseCommandLine (fpSerial, argc,
                              &lSerialNum,
                              &lAddress, &lIncr);

      //if there was no error with the command line
                        arguments
      //calculate next serial number and generate
        serial number file if (rtn == 0)
      {
            lNextSerialNum = lSerialNum + lIncr;

            GenSerialFile (fpSerial, lAddress,
                           lSerialNum,
                           lNextSerialNum);
      }
      //Close the serialization file
      fclose (fpSerial);
      return 0;
}

/*
      ParseCommandLine () - - Parse the command line
                               arguments to get the
                               serial number
                               information
*/
int ParseCommandLine (FILE *fp, int argc, char
                      **argv, long *plSerialNum,
                      long *plAddress, long
                      *plIncr)

{

      int I, nLength, nSerNumberFound=0;

      //Initialize values
      *plSerialNum = 0;
      *plAddress = DEFAULT_ADDRESS;
      *plIncr = DEFAULT_INCR;

      //Loop through command line arguments and
      //process relevant command line options
      for (i=1, i,argc; i++)
      {

            nLength = strlen(argv[i]);
            if ((*(argv[i]) = = ‘-‘) && (nLength >
                                                    2
```

```
                                                  )
                                                  )
        {
              switch ( *(argv[i] +1))
        {
              case 'N':
  nSerNumberFound = 1;
                    sscanf(argv[i] + 2, "%ld",
                                                  p
                                                  l
                                                  S
                                                  e
                                                  r
                                                  i
                                                  a
                                                  l
                                                  N
                                                  u
                                                  m
                                                  )
                                                  ;
                    break;

              case 'I':
                    sscanf(argv[i] + 2, "%ld",
                                                  p
                                                  l
                                                  I
                                                  n
                                                  c
                                                  r
                                                  )
                                                  ;
                    break;

              case 'A':
                    sscanf(argv[i] + 2, "%ld",
                                                  p
                                                  l
                                                  A
                                                  d
                                                  d
                                                  r
                                                  e
                                                  s
                                                  s
                                                  )
                                                  ;
```

```
                    break;

               default:
                    break;

               }
          }
     }

     //Return an error if there was no –N<serial
     number> in the command line
     if (! (nSerNumberFound))
     {
          fprintf (fp, “Tos:Serial number not
passed to serialization program\n”);
          return 1;

     }
     return 0;
}

/*
     GenSerialFile() - -  Generate the serial
number file
*/
void GenSerialFile (FILE *fp, long lAddress, long
                                        lSerialNum
                                        , long
                                        lNextSeria
                                        lNum)
{

     fprint(fp, “T01:%ld\n”, lSerialNum);
     fprint(fp, “T02:%ld\n”, lNextSerialNum);
     fprint(fp, “T03:99\n”);
     fprint(fp, “T04:\n”);

     //Write the formatted data portion
     WriteDataInFormat (fp, lAddress, (char*)
                                        (&lSerialN
                                        um),
                                        sizeof
                                        (long));

}

//Variables for the writing formatted data static
unsigned csum; checksum;
          //line
```

```
static char hex[16]=”0123456789ABCDEF”;

//Record size
#define REC_SIZE 32

//Functions for writing the formatted data

/*

      hexputcsum() - - Convert the number to hex and
                       add the number to the
                       checksum

*/
static void hexputcsum(FILE *fp, int c)
{
      c=c&255
      csum+=c;
      fputc(hex[c>>4], fp);
      fputc(hex[c&15], fp);

}

/*

      putcr() - - Put a carriage return at the end
of the line
*/
static void near putcr(FILE *fp)
{
      fputc(‘\r’,fp);
      fputc(‘\n’,fp);
}

/*
      EmitLine() - - Write a line of data to the
file
*/
static void EmitLine(FILE *fp, int size, unsigned
                     addr, int type unsigned char *
                     data)
{
      int i;

      csum=0
      fputc(‘:’,fp);
      hexputcsum(fp, size);
      hexputcsum(fp, addr>>8);
```

```
      hexputcsum(fp, addr);
      hexputcsum(fp, type);
      for (i=o;i<size;i++)
            hexputcsum(fp, data[i]);
      hexputcsum(fp, 0-csum);
      putcr(fp);
}

/*
      WriteDataInFormat() – - Write the formatted
                                data into the file
*/
void WriteDataInFormat (FILE *fp, long lAddress,
                        unsigned char *szData, int
                        nDataLen)
{
      int c;
      int i;
      int nIndex;
      long lExtAddr=0;      //what the receiver
                            perceives the extended
                            address to be

      nIndex = 0

      //step through the data creating 32 byte
      records while (nDataLen)   {

            unsigned char line_buf[REC_SIZE];

            //make sure the receiver knows the
                        correct address
            if ((lAddress-lExtAddr) & ~65535)     {

                  //We must now send an extended
                  address record if (lAddress >=
                  (1L<<20)) {

                  // 32-bit address record
                  line_buf[0]=lAddress/65536L/256L;
                  line_buf[1]lAddress/65536L;
                  EmitLine(fp, 2, 0, 4, line_buf);
            } else {
                  //20-bit address record
                  line_buf[0]=lAndress/16L/256L;
                  line_buf[0] &= 0xf0;
                  line_buf[1]=0;
                  EmitLine(fp, 2, 0, 2, line_buf);
              }
```

```
                1ExtAddr = lAddress & ~65535;
        }
        //Fill the line_buf[] with data & write it out
        i = nDataLen<REC_SIZE ? nDataLen : REC_SIZE;
        memcpy(line_buf, &szData[nIndex], i);
        EmitLine(fp, i, (unsigned)(lAddress&65535), 0,
line_buf);
    }
    EmitLine(fp, 0, 0, 1, NULL); //end record
}
```

The range, if the range addresses change, error

CHAPTER 11
TEST VECTORS

DEFINITION

A test vector is an array of characters, one character for each pin on the chip, that specify test conditions and expected test results for the chip. If test vectors are stored in a JEDEC or POF file, they will be loaded into the vector buffer when the file is loaded. Test vectors may be examined and modified with the *Buffer/Vectors* command.

Test vectors let the designer verify that the PLD behaves correctly without having to prototype a circuit. A properly designed set of vectors will also ensure that the programmed part is functioning correctly. Most PLD development software will help you generate valid test vectors automatically.

During the vector test, the programmer applies high and low signals to the input pins of a functioning PLD and observes the output pins. The output results are compared to the expected results from the test vectors. Any differences will show up in error messages.

CHARACTERS

The following are valid characters for test vectors:

0	*Apply Vil to an input pin*
1	*Apply Vih to an input pin*
C	*Clock an input pin (Vil,Vih,Vil)*
K	*Clock an inverted input pin (Vih,Vil,Vih)*
X	*Don't care: see* Device/Options *command*
F	*Float the pin (high-impedance)*
N	*This pin is not tested (used for power supplies)*
H	*Expected result on output pin is Vih*
L	*Expected result on output pin is Vil*
?	*Read an output pin and replace this ? character with H, L, or Z*
#	*Randomly generate a 1 or 0 to replace the # in this vector*

ENHANCEMENTS

The **?** and **#** characters are not supported by standard JEDEC files. These two characters let you generate test vectors for any chip by placing a **#** in the column for each input pin, and a **?** on each output pin. When the test is performed the first time (on a known good chip), input conditions (**0**,**1**) are generated at random and the **#** character is replaced by the generated condition. Outputs specified by **?** are read from the device under test and the character is replaced by either **H**, **L**, or **Z**. The next time a test is performed with the same vectors, the pin will be tested and expected to produce the same result as the first chip. This test is not guaranteed to find every fault, so be sure to use lots of vectors.

CHAPTER 12
ERASING EPROMS

METHOD

In order to clear data in an EPROM (in preparation for programming), the chip is exposed to short-wave ultraviolet light, which is typically provided by an EPROM eraser. The UV light penetrates a quartz window on the top of the package, erasing the data. A certain dosage is required to fully erase the part, so brighter light sources erase the part more quickly. Most erasers require between 3 and 30 minutes to erase an EPROM.

EPROMs are usually erased using a mercury vapor bulb emitting 2537 Angstroms. The bulbs most commonly used are like fluorescent tubes, but without the phosphor; they are often called germicidal bulbs. The ultraviolet light can be harmful to eyes, so erasers are equipped with shields to prevent light leakage.

PROCEDURE

Erasing an EPROM properly is not as simple as it appears. It is possible to have a partially erased chip that appears to be blank but is not blank when read at a different voltage or temperature. Use this procedure to determine a safe erasing time:

- Start with a programmed chip.
- Erase the part in one minute increments and use the *Device/Blank* command to test it each minute.
- Once the programmer says the chip is blank, double the erase time to give yourself an adequate safety margin.

Some types of parts take longer to erase than others. You may need to experiment with the various parts you use. An EPROM based part with a security bit feature (a PLD or microcontroller) is designed so that the security address will typically be the last bit to erase.

☞ The adhesive used on labels often blocks UV light. If the chip erases slowly, try cleaning the window with alcohol or a stronger solvent .

Sunlight and fluorescent light can erase chips; however, it usually takes months or years. You should cover the window with an opaque label to make the data permanent.

Some EPROM based parts are available in inexpensive plastic packages. These parts can't be erased because they have no window. These chips are referred to as one time programmable (OTP) EPROMs.

CHAPTER 13
EMULATION MODES

16V8 AND 20V8 ARCHITECTURE

GAL stands for **G**eneric **A**rray **L**ogic and was invented by Lattice Semiconductor as a method of replacing many standard PAL arhcitectures with a single general purpose architecture. The GAL16V8 (20-pin DIP) and 20V8 (24-pin DIP) are EEPROM based second generation PAL devices. Each of the eight output pins uses an output logic macrocell (OLM) that can be configured to be combinatorial or registered, active high or active low, and can have an output enable (OE) term. This architecture is a superset of the standard PAL devices. Furthermore, the GALs fuse map is also a superset of the standard PAL.

PROGRAMMING

The programming algorithm can automatically reconfigure these two GAL devices to emulate 42 different 20- and 24-pin standard PALs. These emulation modes accept standard PAL JEDEC files. For example, you can program, verify, and test the GAL16V8 using the "GAL16V8 as 16L8" algorithm and a JEDEC file for a PAL16L8. The programmed part can be used in place of a PAL16L8.

20XV10 ARCHITECTURE

Lattice has introduced another GAL device called a GAL20XV10 which can emulate the standard PAL20L10, PAL20X10, PAL20X8, and PAL20X4 architectures in the same manner as described above for 16V8s.

EMULATION CONSIDERATIONS

All GAL devices are programmed using generic algorithms; that is, the programmer reads the part you put in the socket to determine the correct programming voltage, timing, etc. That way, you don't need to worry about speed and power suffixes on the device you are programming.

GALs are normally guaranteed to reprogram up to 100 times which is great for prototype design efforts. GAL devices are available from Lattice Semiconductor, National Semiconductor, SGS-Thomson, and VLSI Technology. AMD's PALCE16V8 and PALCE20V8 are 100% JEDEC compatible with the original GALs; thus, they support all the same emulation modes.

AMD also has a 24-pin PALCE16V8HD that is not plug-in compatible to the standard 16V8, but will accept all the same JEDEC files for its emulation modes.

CHAPTER 14
TROUBLESHOOTING AND MAINTENANCE

This chapter provides information on troubleshooting and maintenance for the BP-2000 Series Programmers including FAQ's, Common problems and solutions, and error/warning messages. Contact information and instructions for downloading from the BBS are also found herein.

CUSTOMER SERVICE

WHEN YOU NEED HELP

The information in this chapter may help you solve or identify a problem with your programmer. If you have a problem that you cannot solve, please call us. We are dedicated to making BP Microsystems programmers as trouble-free as possible.

HOW TO REACH US

Technical Support—	800-225-2102
	713-688-4600 (Outside the U.S.)
	713-688-0920 (Fax)
	tech@bpmicro.com (email)
	4100support@bpmicro.com (email)
Bulletin Board—	713-688-9283
Sales Support—	713-688-4600
	info@bpmicro.com (email)
Web Page—	www.bpmicro.com (Internet)

SOFTWARE UPDATES

The control software for your programmer is updated on a frequent basis (typically each six weeks) to add features and provide you with support for new chips.

Software updates may be obtained from BP Microsystems as a subscription, or by downloading from our electronic bulletin board system (BBS). Please contact a sales person to purchase a software subscription.

Your programmer is designed to be highly flexible and programmable, allowing it to program a wide variety of chips. Consequently, when a problem does arise, it can usually be fixed with just a software update.

CALLING THE TECHNICAL SUPPORT LINE

You can obtain technical support from BP Microsystems by calling 1-800-225-2102 (or 713-688-4600 outside the U.S.). You can also reach us by fax at 713-688-0920 or send email to tech@bpmicro.com. We request that you have the following information ready when you contact us:

- The model number of the BP Microsystems programmer (bottom of screen)
- Your software version number (from the top of the screen)
- The exact error message and error number you received
- The exact algorithm that was selected (bottom of screen)
- The exact part number on the chip you were trying to program
- The command you executed
- The results of running the <**Alt-D**> Self-Test command on your programmer

It is also useful to have a print-screen of the error, or to print out a log of the error (see *Info/Log* command). You may be asked to upload your file and/or send in your devices so we can analyze the error at the factory.

☐ *If you need to return your programmer to BP Microsystems for any reason, you must call and get a Return Material Authorization (RMA) number before shipping; mark the RMA number clearly on the shipping container. Be sure to include a description of the problem experienced, a return address, contact person and a phone number.*

SOFTWARE UPDATES

The control software for your programmer is updated on a frequent basis (typically each six weeks) to add features and provide you with support for new chips.

Software updates may be obtained from BP Microsystems as a subscription, by downloading from our electronic bulletin board system (BBS) or via our web site (www.bpmicro.com). Please contact a sales person to purchase a software subscription.

Your programmer is designed to be highly flexible and programmable, allowing it to program a wide variety of chips. Consequently, when a problem does arise, it can usually be fixed with just a software update.

Our policy is to resolve user issues as quickly as possible, often within a day or two, and to release update software immediately on the BBS and website. Therefore, if you have encountered a software issue, there is a good chance that it has already been fixed and placed on the BBS and website.

We recommend that you obtain the latest software revision before calling our support line with a software problem. Many of our technical support calls result in the user obtaining the latest version of the software.

DOWNLOADING FROM THE BBS

You can access our electronic BBS to obtain a software update by calling 713-688-9283 (in the U.S.) The system requires 8 data bits, 1 stop bit, and no parity. Once you have logged onto the system, it will give you instructions on how to download the latest software. See the *Info/BBS* command for more details on the BBS operation.

If your modem supports speeds faster than 9,600 baud, you may need to call a different telephone number to utilize the fastest speed. Once you have logged on, you can read a list of telephone numbers on the BBS that will allow the fastest transfer rates.

You will need a communications program to use your modem. The communications program allows your computer to access our BBS and save a file to your disk. You will need to select the same download protocol on both your communication device and our BBS. Recommended protocols are Zmodem and Ymodem because they provide the fastest transfer and easiest operation. The file you download will be a self-extracting compressed file. Once you have downloaded the file, execute it to extract its contents. You may then delete the file you have downloaded – it will not be needed again.

TESTING THE HARDWARE

The programmer can test its own hardware quite extensively. The Self-Test routine can detect problems in the pin-drivers, power supply, microprocessor, data cable, printer port, and several other circuits. The hardware test cannot detect problems resulting from a dirty socket (see below). To execute the test:

- *Remove any chips from the programmer sites*
- *Press <**Alt-D**> hot-key*
- *Choose to test a single unit or all units*
- *Watch the screen for any error messages*

- *If you receive an error during the test, please call Technical Support for assistance*

ERRORS WHILE PROGRAMMING

If you experience problems while trying to program a chip, try to narrow down the problem. If you receive a **Cannot program** or **Cannot erase** error while programming:

- Make sure you have selected the proper programming algorithm
- If you are using a PLD or microcontroller, the device may have been previously secured
- The device may not be fully erased (this should be verified by the *Blank check before programming*: option, if you have not turned it off)
- The device may have a newer die than the one supported by the programming algorithm

You may want to run the *Device/Compare* command to see where the command failed. Look at the first error indicated by the *Compare* command. This is the location that caused the *Device/Program* command to abort and return an error message. Three possibilities exist: 1) the byte or bit didn't program at all, 2) there are extra programmed bits, or 3) a combination of unprogrammed bits and extra programmed bits. On memory devices, compare the two hex data values to see what went wrong. On most EPROMs, the erased state is FFh and the fully programmed state is 00h.

If the expected value has more 0 bits than the value read from the chip, the chip may not have been fully erased before programming. If the byte read from the chip is completely unprogrammed (FFh), you are probably either using the wrong algorithm or have a defective chip. If you find that the same bit fails every byte, you should suspect a dirty socket or a bent IC pin.

You can expect a small number of fuse-link programmable PLDs to fail during programming. This is because they cannot be fully tested at the factory without blowing out the fuses! Most semiconductor manufacturers claim you should get a 98% programming yield. Your yield shouldn't drop below the value specified by the semiconductor manufacturer.

Most semiconductor manufacturers fully test EPROM and EEPROM-based PLDs. You should achieve a very high yield on these parts. These parts do, however, have a limited number of erase cycles. Most parts can be erased and programmed up to 100 times without failure.

CLEANING A DIRTY DIP SOCKET

If the DIP socket becomes dirty, it will fail to make contact with all the chip pins. The simple fix is to place your chip in the socket, push the lever down, and slide the chip left and right a few times. Cleaning sockets with a blast of high-pressure air on a regular basis is also recommended.

If this does not resolve the problem, run the hardware test described above. If your hardware has passed the Self-Test, there may be an error in the programming algorithm you are using or the semiconductor manufacturer may have updated the programming algorithm for your device. In either event, your problem will probably be corrected with a software update.

PLD VECTOR TEST ERRORS

If you get consistent error messages during vector tests, refer to the *Device/Options* command. You may need to use a different X-value. If you get an "*Excessive current detected*" error, your test vectors may be applying conflicting signals to the chip's output pins; you should correct the vectors. See "*Error 38: Functional test failed*" and the section on *Test Vectors* in this chapter for other possible causes of vector failure.

POWER-ON SELF-TEST (POST)

When power is applied to the programmer, it performs a power-on Self-Test (POST). This test checks RAM, ROM, CPU, analog circuits, and basic system integrity.

- **Do not attempt any programming operations until the POST is complete.**

If the POST fails, the red ERROR LED will be on. Failure codes are:

3 short flashes	Cannot Self-Calibrate
2 short flashes	ROM checksum error
1 short, 1 long flash	RAM error

COMMON PROBLEMS & SOLUTIONS

The following sections describe the most common problems with operation of the BP-2000 Series Programmers, along with troubleshooting tips and solutions.

UPGRADING THE BP SOFTWARE

When upgrading the software with an upgrade code, three things can happen.

1. The software will produce a message reading *upgrading* and complete the upgrade. At which time the customer needs to turn off the unit and exit software and reboot for code to become effective.

2. The software can give an "*Invalid Code*" error, which means the code was entered wrong or the serial for the code is different. The codes are serial number dependent. The Self-Test can be run on the site to determine the serial number. Contact Technical Support for further assistance, page 14-1.

3. The software can give an "*EEPROM Contents Invalid - No end record*" error. This can happen if the software is interrupted during the upgrade process. If this should happen, contact Technical Support, page 14-1.

DISABLE SCREENSAVER ON DIGITAL SWITCHBOX

Use this procedure to disable the screensaver on the digital softserv switchbox.

Make sure that the keyboard is active for the screen you are in. It doesn't matter which computer screen. If the keyboard happens to be locked up, disconnect and reconnect it. The press this key sequence:

<**Left CTRL**>, then <**v**>, then <**0**> (zero), and <**Enter**>. Keystrokes must occur within 2 seconds of each other.

This procedure will have to be done each time the BP-4100/4500 is powered up unless you save this new setting with the Keep command:

<**Left CTRL**>, then <**k**>.

ERRORS AND WARNINGS

If you experience problems while trying to program a chip, try to narrow down the problem. If you receive a "***Cannot Program***" or "***Cannot Erase***" error while programming:

- Make sure you have selected the proper programming algorithm
- If you are using a PLD or microcontroller, the device may have been previously secured
- The device may not be fully erased (this should be verified by the *Blank check before programming*: option, if you have not turned it off).
- The device may have a newer die than the one supported by the programming algorithm.

You may want to run the *Device/Compare* command to see where the command failed. Look at the first error indicated by the *Compare* command. This is the location that caused the *Device/Program* command to abort and return an error message. Three possibilities exist:

1) the byte or bit didn't program at all,
2) there are extra programmed bits, or
3) a combination of unprogrammed bits and extra programmed bits.

On memory devices, compare the two hex data values to see what went wrong. On most EPROMs, the erased state is FF and the fully programmed state is 00.

If the expected value has more 0 bits than the value read from the chip, the chip may not have been fully erased before programming. If the byte read from the chip is completely unprogrammed (FF), you are probably either using the wrong algorithm or have a defective chip. If you find that the same bit fails every byte, you should suspect a dirty socket or a bent IC pin.

You can expect a small number of fuse-link programmable PLDs to fail during programming. This is because they cannot be fully tested at the factory without blowing out the fuses! Most semiconductor manufacturers claim you should get a 98% programming yield. Your yield shouldn't drop below the value specified by the semiconductor manufacturer.

Most semiconductor manufacturers fully test EPROM and EEPROM-based PLDs. You should achieve a very high yield on these parts. These parts do, however, have a limited number of erase cycles. Most parts can be erased and programmed up to 100 times without failure.

ERROR MESSAGES

Error 3: Cannot reset hardware

The software cannot establish communications with the programmer. Here are some suggestions:

- Be sure the programmer has proper power and that the power LED is on.
- Make sure the cable from the programmer to the computer is properly connected to a parallel printer port. If you are using a ribbon cable, this is probably the problem (ribbon cable connectors are designed for use inside a chassis where the cable is not flexed). You should use a shielded 25 conductor cable (not an RS-232 cable).
- The programmer may be damaged. Try another computer and/or parallel port and see if it works. If not, see *Calling the technical support Hotline* in this chapter.

Error 4: Excessive current detected.

- The protection circuit has shut off the power.
- The command was aborted to protect the programmer and hopefully not damage the chip. The part was taking too much current from the programmer. Possible causes are:
- The part may be inserted backwards and the continuity test has been turned off or did not successfully detect the device.
- The wrong algorithm could be selected and improper voltages were applied to the chip in the programmer site.
- If you are running a vector test, your test vectors may be applying conflicting signals to the chip's output pins. You should correct the vectors.
- You may want to remove the chip and run the hardware test, <**Alt-D**>, to make sure all the pin drivers are functioning correctly. If the hardware passes the test, be sure you have the correct algorithm (device entry) selected for your part. If the error still occurs and you are sure the device is inserted correctly, then you should suspect a faulty chip.

Error 5: Hardware time-out.

This error message is generated when the software was waiting on a response from the programmer while executing a command and the programmer did not respond within the expected amount of time. This error may result from

several causes. You may be experiencing communication errors (see "*Error 3: Cannot reset hardware*" above). There may be a bug in the software for this particular algorithm (see *Error 10: Error in programming algorithm*" below. See also *Power-on Self-Test* above).

Error 6: Wrong model number.

See "*Error 3: Cannot reset hardware*" above for possible causes.

Error 8: LPTx: is not a functioning port.

The parallel port LPTx (where x=1, 2, or 3) that is selected with the *Configure* command does not exist in your computer, is not functional, or has a bad cable connected to it.

Error 9: Programmer execution error.

The programmer failed an internal consistency check. See "*Error 3: Cannot reset hardware*" and "*Error 5: Hardware time-out*" above for possible causes.

Error 10: Error in programming algorithm. Please call technical support.

The software has detected an internal error. You should contact BP Microsystems to report the error. You may need to obtain a software update. See *Calling the technical support Hotline* in this chapter.

Error 11: There is no data in the buffer. You must load a file or read a chip.

A command tried to read data from the buffer to program or verify a chip, but nothing has been loaded into the buffer yet or the buffer was recently cleared.

Error 14: There is no chip in the programmer site.

Be certain that your chip is inserted correctly. If the chip was inserted correctly, remove it and run the hardware Self-Test to be sure your programmer is functioning correctly, <**Alt**-**D**>. A defective chip may cause this error. When using an Autohandler, the contactor may not have closed or the connection between the programmer and the contactor may be disconnected.

Error 15: The chip is not inserted in the programmer site correctly.

The continuity test determined that the chip in the programmer site does not have continuity on all the proper pins. You should examine these pins carefully. Possible causes are:

- A bent pin.
- The chip is not in the proper position in the programmer site.
- The algorithm selected has a '*', indicating it requires an adapter, but you did not use the adapter, or vice-versa.
- The socket is dirty and not making a connection. See *Cleaning a Dirty Socket* in this chapter.
- The device may be a very low power device that is not properly detected by our continuity methodology. If so, please let us know.

Error 16: The chip is inserted backwards.

The chip has passed the continuity tests, but appears to have the GND and Vcc pins improperly placed in the socket. If the DIP, SOIC, or TSOP device is not actually inserted backwards and the LCC, PLCC, or QFP is not accidentally rotated, then the device is probably defective. Try a known good device.

Error 17: Out of base memory. You should have at least 200K free.

Your computer's configuration does not have enough RAM available to run the software. You should have 640K RAM installed with at least 200K available for program execution. Memory resident programs, such as network drivers, may reduce the RAM available to the programmer, so you may need to remove these programs from your CONFIG.SYS and AUTOEXEC.BAT files. If you are using DOS 5.00, you can specify that DOS be loaded into high memory, saving base memory for BP software. See your DOS manual for details. The *mem* or *chkdsk* command will show you how much conventional memory is available.

Error 18: Temporary file error.

Our software's virtual memory manager is trying to store data that is currently not needed in RAM to the disk. The program was unable to create a temporary file or the disk is full. You should make sure you have plenty of disk space (the larger the data files, the larger the requirement for temporary disk space) and set the DOS environment variable TMP to point to the

directory you wish to use for swap space. The program does take advantage of EMS memory if you have an expanded memory manager installed. This is much faster than using the disk for temporary swap space. If you want to specify that your hard disk (C:) can be used for temporary file storage, then execute the following command in your AUTOEXEC.BAT file:

SET TMP=C:\

Error 21: Cannot program.

Not able to program the device in the programmer site. See *Errors While Programming* in this chapter.

Error 22: Cannot erase.

Not able to erase the device in the programmer site. See *Errors While Programming* in this chapter.

Error 23: Invalid electronic signature in chip (device ID).

The chip may be damaged or the chip manufacturer may have changed the programming algorithm without notifying us. See "*Error 25: Invalid electronic signature in chip (manufacturer ID)*" below.

Error 24: Invalid electronic signature in chip (algorithm ID).

The chip may be damaged or the chip manufacturer may have changed the programming algorithm without notifying us. See "*Error 25: Invalid electronic signature in chip (manufacturer ID)*" below.

Error 25: Invalid electronic signature in chip (manufacturer ID).

Many of the new EEPROM based PLDs (such as all the GALs) have electronic identifiers that specify the manufacturer, the device code, and the proper programming parameters. The most common cause of this error is when you have selected the wrong manufacturer for the particular part you are using (*e.g.*, you may have a National Semiconductor part in the programmer site and a Lattice part selected). It is also possible that your chip has a newer ID than your software revision supports. See *Software updates* in this chapter.

Error 26: Device is not blank.

The *Device/Blank* command was executed or the "*Blank check before programming:*" option was enabled in the *Device/Options* dialog box and the device in the programmer site is determined to have programmed data. Possible causes are:

- The part needs to be erased longer.
- The device was previously programmed and cannot be erased (OTP EPROMs and fuse-link PLDs).
- The wrong algorithm was used.

Error 27: Device is not secured.

An attempt to secure a device was made, but it failed. See *Errors While Programming* in this chapter.

Error 28: Data in chip does not match buffer.

A verify operation was performed to see if the chip and the buffer have the same data, but there is a difference in the two patterns. The verify may have been performed via the *Device/Verify* command or the *Device/Program* command when the "*Verify after programming*" was enabled under the *Device/Options* dialog box. The verify will not show where the difference occurred but the *Device/Compare* command will show you all the discrepancies. Here are some reasons why a device may fail to verify:

- The part was secured.
- The part was programmed with a different pattern.
- The part has not yet been programmed.
- The device was incorrectly programmed by a different system.
- The wrong algorithm is selected.
- The chip was not properly erased before programming (this would be caught by the blank check if it was enabled).
- The device is not inserted correctly and you have the continuity test turned off or you ignored its error messages.
- The device is defective.

Error 31: Database file is invalid. The .EXE file is corrupted.

The .EXE file you are executing has been corrupted. You should get a new copy from BP Microsystems. See *Calling the technical support Hotline* in this chapter.

Error 32: Sorry, algorithm not found. Please call technical support.

The .EXE file you are executing has been corrupted. You should get a new copy from BP Microsystems. See *Calling the technical support Hotline* in this chapter.

Error 33: You must reselect the chip you want to program.

The device was selected before establishing communications with the programmers, perhaps prior to turning on the programmer or before switching to a different programmer. Simply reselect the chip and you will be in business again.

Error 36: You must properly install the correct socket module.

The software interrogates the socket module before each operation to determine the correct mapping for the algorithm selected. You will get this error if:

- There is no socket module installed.
- The socket module installed does not support the device you have selected (*e.g.*, you have selected a 20 pin device and you have a 28 pin PLCC socket module attached).
- The socket module installed is not supported by the version of the software you are using. Use the latest version.
- The pinout has not yet been defined for this package type. It may be an oversight on our part. If so, please call technical support and inform us of this problem.

Error 37: Illegal bit detected.

If the *Device/Options* dialog box has the "*Blank check before programming:*" set to **ILLEGAL_BIT**, then you will get this message if the blank check fails. This message indicates that a bit was programmed which should not have been programmed. You may not program the pattern in the buffer on top of the pattern that is in the chip. See "*Error 28: Data in chip does not match buffer*" for possible causes.

Error 38: Functional test failed.

The *Device/Test* command was executed or *test after verify* was enabled in the *Device/Options* command and the device did not perform the way the test

vectors expected. For more details, see *Test Vector Troubleshooting* at the end of this chapter.

See PLD Vector Test Errors at the beginning of this chapter for other common explanations for vector failures.

Error 39: Device already secured.

The device cannot be legitimately programmed, read, etc., because it has been secured. If it is a PLD it may still be functionally tested with the *Device/Test* command.

Error 40: No test vectors present.

The file you loaded did not contain any test vectors. Therefore, the *Device/Test* command will not be executed. You should set the *Device/Option* "Vector test after verify" to NONE.

Error 41: Error reading file.

The *Buffer/Load* command was executed inside a macro file and the buffer could not be loaded. This error message is not displayed on the screen, but is returned to DOS when the software is being run via a batch file.

Error 42: Error in writing file.

The *Buffer/Save* command was executed inside a macro file and the buffer could not be saved. This error message is not displayed on the screen, but is returned to DOS when the software is being run via a batch file.

Error 43: Error in macro file.

A macro file was being played back and an error was detected in the syntax of the file. Possible causes are:

- The macro file is corrupted.
- The macro file was recorded with an earlier version (<V2.00) of the software.
- The macro file was generated by a user's application or text editor and does not conform to the proper macro file format.

Error 44: Internal error. Please call technical support.

The software detected an internal inconsistency. This may be caused by the computer not performing correctly.

Error 45: Hardware requires calibration. Please call technical support.

The Self-Test, <**Alt-D**>, has detected that the hardware is improperly calibrated. The unit must be returned for repair. See *Calling the technical support Hotline* in this chapter.

Error 46: AFS software required to execute this function.

This is a function that is available to users that have purchased the Advanced Feature Software only. In order to use the chosen function you must buy the AFS upgrade. See *Calling the technical support Hotline* in this chapter.

Error 47: Self-Test failed. This unit may need service. Please call technical support.

The Self-Test, <**Alt-D**>, has detected a hardware problem. The unit may need to be returned for repair. Note the exact error message and see *Calling the technical support Hotline* in this chapter.

Error 48: Cannot Unprotect.

An attempt was made to unprotect a sector and failed. See *Errors while programming* in this chapter.

Error 49: Cannot Protect.

An attempt was made to protect a sector and failed. See *Errors while programming* in this chapter.

Error 50: Device sum does not match sum specified in AFS/Options.

The sum calculated on the device does not match the sum entered in the *AFS/Options Checksum Verify* command. Check this option to see if a mistake was made when entering the sum value. Also check the buffer checksum to see if it matches the value entered for *Checksum Verify* or if any data in the buffer has changed.

Error 51: Maximum failures reached.

The maximum failures entered in the *Device/Handler* menu has been reached and therefore programming of the current job has been stopped.

Error 52: DynCall Stack Underflow.

The internal dynamic linker underflowed its reference table. If this error reoccurs, then call BP Microsystems Technical Support Line.

Error 53: DynCall Stack Overflow.

The internal dynamic linker overflowed its reference table. If this error reoccurs, then call BP Microsystems Technical Support Line.

Error 57: You must purchase support for this device to use it.

The device that you selected is not supported in the default device set for this programmer. Call BP Microsystems Sales line to purchase an upgrade code for your programmer.

Error 60: The demo period for this programmer has expired.

This programmer is a demo from BP Microsystems and the demo period has expired. Call BP Microsystems Sales Department for an upgrade code to extend the Demo period.

Error 61: Concurrent programmer did not initialize properly.

The programmer did not initialize correctly. Cycle the power on the programmer and try your operation again. If you continue to get this error message, send the programmer in for repairs.

Error 65: Concurrent Unit has the wrong socket module.

The specified programming site does not have the same socket module as the master site. The site must contain the same socket module as the master programmer in order to program parts on that site. The site has been temporarily disabled. Starting a new device operation with the correct socket module on the site will re-enable the site.

Error 66: Concurrent unit has the wrong technology adapter.

The specified programming site has the wrong technology adapter (TA). Cycle the power on the programmer. If the error persists, call BP Microsystems Technical Support.

Error 67: Concurrent unit has the wrong BIOS.

The specified programming site has the wrong BIOS. Cycle the power on the programmer. If the error persists, call BP Microsystems Technical Support.

Error 68: Concurrent unit has the wrong number of pin drivers.

The specified programming site has the wrong number of pin drivers. Cycle the power on the programmer. If the error persists, call BP Microsystems Technical Support.

Error 69: Concurrent unit is not available.

The specified programming site is not responding to commands. Verify that the programmer number is correct. Cycle the power on the programmer and try again. If the error persists, call BP Microsystems Technical Support.

Error 70: The buffer data cannot be used to program this device.

You loaded a file type that is not a valid option for the currently selected device. Re-select the device and load the buffer again. If the error persists, call BP Microsystems Technical Support.

WARNING MESSAGES

Warning: Device code is not correct

The programmer automatically checks the electronic identifier of your memory chip; however, this feature may be disabled in the *Device/Options* command. If your chip does not read out the identifier expected by the software, two possible actions may be taken by the programmer. If you have "*Check electronic identifiers:*" set to AUTO-SELECT and your chip reads out a known identifier, the software will automatically select the new algorithm for your chip. If you have it set to **ENABLE**, you will receive an "*Abort, Retry, Ignore or Select?"* query. The **Select** option will try to select the appropriate algorithm. Use it if you feel the wrong algorithm may have been selected in the first place. You can replace the chip and use **Retry** or you can abort the command. Finally, you can ignore the identifier and proceed.

Occasionally a semiconductor manufacturer will change the die on a chip to add the identifier. If this happens, the software may expect the identifier to be present in your chip. If you are using an old chip made before the identifier was added, you will get an error message. It is safe in this case to ignore the warning message. You can use the *Device/Options* command to disable the identifier altogether if you want to.

Warning: Device is not blank

You will get this warning when using the *Device/Program* command with the "*Blank check before programming*" operation enabled in the *Device/Options* dialog box. You are given the option to "*Abort, Retry, or Ignore*", which allows you to try another chip before proceeding or to go ahead and program data on top of the data that may already be in the chip.

Warning: Device has been secured

You will get this message only on devices that have the ability to read the security bit prior to performing any other operation. Thus, you may get this when trying to read some PLDs and some microcontrollers. You are given the option to "*Abort, Retry, or Ignore*", which allows you to try another chip before proceeding or to go ahead and read the erroneous data from the chip.

Warning: X fuses in buffer and Y fuses in chip

You will get this message when you try to program a PLD that has a different number of fuses than the number of fuses loaded in the buffer. This is a precautionary measure designed to help you keep from accidentally programming the wrong JEDEC file into a chip.

This message may not always be the best advice. For example, if you have a GAL22V10 (5892 fuses) selected and a PAL22V10 (5828 fuses) JEDEC file loaded. It is okay to program this JEDEC file into the GAL22V10 because they are 100% architecturally compatible; however, the GAL22V10 has 64 additional fuses allocated as the UES (see *Device/UES* command for more details). If you ignore this message on the first operation, you will not receive any more warnings on the subsequent programming operations.

TEST VECTORS

If you get consistent error messages during vector tests, refer to the *Device/Options* command. You may need to use a different X-value. If you get an "*Excessive current detected*" error, your test vectors may be applying conflicting signals to the chip's output pins; you should correct the vectors. See "*Error 38: Functional test failed*" for other possible causes of vector failure.

Test vectors allow a user to power up and operate the device under test (DUT) to simulate operation in the user's circuit. Power is applied, high and low voltages are applied to the DUT's inputs, and its outputs are observed. Each test vector describes what inputs to apply to the device (**1, 0, F, C, K, U, D**), which outputs to examine (**H, L, Z**), and which pins to ignore (**X, N**).

A test vector is an array of characters, one character for each pin on the chip, that specify test conditions and expected test results for the chip. If test vectors are stored in a JEDEC or POF file, they will be loaded into the vector buffer when the file is loaded. Test vectors may be examined and modified with the *Buffer/Vectors* command.

Test vectors let the designer verify that the PLD behaves correctly without having to prototype a circuit. A properly designed set of vectors will also ensure that the programmed part is functioning correctly. Most PLD development software will help you generate valid test vectors automatically.

During the vector test, the programmer applies high and low signals to the input pins of a functioning PLD and observes the output pins. The output results are compared to the expected results from the test vectors. Any differences will show up in error messages.

The following are valid characters for test vectors:

0	Apply Vil to an input pin	**N**	This pin is not tested (used for power supplies)
1	Apply Vih to an input pin	**H**	Expected result on output pin is Vih
C	Clock an input pin (Vil, Vih, Vil)	**L**	Expected result on output pin is Vil
K	Clock an inverted input pin (Vih, Vil, Vih)	**?**	Read an output pin and replace this **?** character with **H**, **L** or **Z**
X	Don't care; see *Device/Options* command	**#**	Randomly generate a 1 or 0 to replace the **#** in this vector
F	Float the pin (high-impedance)		

Table 12 – Valid Test Vector Characters

ENHANCEMENTS

The **?** and **#** characters are not supported by standard JEDEC files. These two characters let you generate test vectors for any chip by placing a **#** in the column for each input pin and a **?** on each output pin.

When the test is performed the first time (on a known good chip) input conditions (**0**,**1**) are generated at random and the **#** character is replaced by either **H**, **L** or **Z**.

The next time a test is performed with the same vectors, the pin will be tested and expected to produce the same result as the first chip. This test is not guaranteed to find every fault, so be sure to use lots of vectors.

TROUBLESHOOTING TEST VECTORS

When a part does not pass test vectors, it may be due to one or more of the following causes:

1. Defective Device - The most obvious reason is that the device itself is defective. This is the main purpose of vector testing. Devices may pass verify but have logic errors that can only be detected in operation. This is the primary reason to run test vectors on each part verified.

2. Incorrect Vectors - The vectors do not describe what the part *should* be doing. Any discrepancies appear as error messages. Either the designer did not simulate the vectors to verify that the vectors were written correctly, or the simulator didn't accurately model how the part will truly perform (a very common problem).

   The simulators that are built into most development systems are unit-delay two-state simulators that don't know the difference between an **X** (unknown value) and a **0**. They also assume that all gates have **1** unit of delay, and that all registers have **0** setup and hold time. These oversimplifications can easily lead to a discrepancy between the vectors and the DUT's behavior.

3. Incorrectly Programmed Device - The DUT is incorrectly programmed. Either the wrong file or the programming algorithm doesn't configure the part correctly, even though it may verify.

   If the part has a second source, try programming the alternate part and running vectors. A difference here may point to a programming problem or a difference between the parts, such as a power-up reset state, asynchronous error, or even a design defect in the silicon.

If another programmer is available, program a part on each programmer and verify each part on the other programmer. Any verify errors indicate a difference in programming algorithms, and a mistake made by one or more of the programmers.

4. Undriven Inputs - The vectors assume that undriven inputs (**X, F, Z, N**) will be high or low. Most simulators assume that all pins marked **X** will be low. The usual case is that these pins are tied to a pull-up resistor (applying a high state to the input, not low). Changing the **X** value in the Options box will change the vector test results, possibly fixing the problem.

   ☐ *Changing the **X** state to **0** causes the programmer to drive all pins marked **X** to **0**, which may cause high currents to flow if the part is driving an **H** on that pin, so be cautious about changing this switch. It is better to tell the simulator that **X** is high in the design software. It is even better to make* no *assumptions about the **X** state.*

5. Ground Bounce - Ground bounce can induce failures. The part may not work correctly in the programmer, even though it may work correctly in circuit. Ground bounce occurs when a clock input changes (**C** or **K**) and that causes several outputs to change state (**H** to **L**, **L** to **H**, **Z** to **L**, or **Z** to **H**). This transition causes large currents to flow throughout the Vcc and GND pins very rapidly, producing a voltage on the die's ground terminal that can produce double clocking.

   Worst-case parts are high-speed (5-15 ns) CMOS parts with many outputs. Usually, ground bounce induces an extra clock pulse, and can be positively identified when the circuit advances to an extra state, such as a counter that rolls over from all **HHHHHHHH** to **LLLLLLLH** instead of **LLLLLLLL**.

6. Preload error - The parts were written with a preload vector, <**B**> or <**P**> and the preload works differently on the programmer than the compiler expected. Some parts don't support preload, so you will get an error message. Sometimes, the device's spec is ambiguous, allowing a different interpretation by the simulator's and programmer's engineers.

7. Power-up Reset error - The part doesn't perform an internal power-up reset, or it is reset to a different state than expected. Failures will typically occur on the first vector that has an **H**, **L** or **Z**. In general, it is not good practice to assume a power-up state either in vectors or in circuit.

All asynchronous circuits with feedback must be initialized before the output will be in a known state. If the circuit uses registers, the user should consult the datasheet on the exact part to verify power-up state because similar parts from different manufacturers may sometimes reset differently (e.g., AMD 22V10 v. TI 22V10).

If the problem persists, it may be because the part is sensitive to slew rate of the Vcc pin. BP programmers raise the Vcc in less than 1µs by default; however, we can change it per algorithm if necessary. Other programmers may use slower default rates.

8. Multiple Clock Pins - Problems may occur with synchronous circuits with multiple clock pins. The vector that fails, or a preceding one, will have two **C** or **K** pins. The designer probably assumed that both clocks will change simultaneously. The programmer can apply two clocks with only a few ns skew, so it will probably not cause problems.

   Some other programmers apply clocks starting with the highest numbered pin, working its way down. Unless you're using an autohandler, it is best to write test vectors with only one clock pin changing at a time to eliminate this problem.

   It is best to avoid simultaneous clocking when possible. Some devices may fail to clock registers simultaneously even when the pins are tied together, due to different internal propagation delays from the pins to the register's clock inputs.

9. Simultaneous Input Changes - Asynchronous circuits with multiple inputs changing simultaneously may cause problems. You will see a vector that differs from the previous vector by having more than one pin change from **1** to **0**, **0** to **1**, **0** to **U**, or **1** to **D**.

   The JEDEC standard says that *no* assumptions should be made about the order of pins being applied to the DUT. The programmer can apply all inputs with very little skew to minimize problems of this sort.

   Any circuit that requires two inputs to change simultaneously is an arbiter, producing a different result when one changes before the other. Therefore, it is better to write two test vectors so the sequence will be known and the output can always be predicted. This allows the programmer to test the DUT, rather than the other way around. Other programmers may apply inputs in a different order, producing different results.

10. Adapter or Connector Problems - Adapters or connections placed between the programmer and the DUT will add inductance,

capacitance and resistance. This causes a degradation of signal quality, causing problems especially on fast parts.

A cable between the programmer and an autohandler or other contactor is always suspect. If this is the case, the problem will go away if a part is placed directly in the programmer site.

Typically, conditions may improve if ground and Vcc traces are made as wide and short as possible, and a 100 nF bypass capacitor is placed at the DUT between Vcc and GND. Also, connect all Vcc pins together and all GND pins together at the DUT.

11. DUT incorrectly connected - Problems can occur due to a bent or dirty pin, using the wrong adapter, an adapter with the wrong pinout, or a part inserted incorrectly in its programmer site. Be wary of specially programmed adapters for autoHandlers, and poorly labeled adapters for QFPs.

12. Race Conditions and Other Timing Faults - Asynchronous circuits may have race conditions or other timing faults. These faults are difficult to detect because they are subtle and require the engineer to carefully check minimum and maximum propagation delays, setup and hold times.

    The circuit will have feedback so it can latch a state. One common symptom is a design that formerly passed vectors but now fails on newer (faster) devices. High-speed devices are more likely to exhibit these problems.

    The best weapon against this sort of fault is INT's PLDLAB90 software. The DUT may be sensitive to slew rates and output loading in this case, causing different results on other programmers.

13. Pattern Sensitive Parts - Some PLDs are pattern sensitive, and this can result in very subtle flaws. They may work correctly in 99% of all designs but fail on a particular fuse pattern. This may escape detection for years but suddenly appear, causing mysterious problems.

    The part may have sensitivity to the way it is being operated. For example, some PLDs have an over-sensitive power-up reset circuit that resets registers when too many outputs change state simultaneously.

GLOSSARY

This glossary has been included for convenience and to assist in grasping an overall understanding for the terminology and jargon used within the software and hardware engineering fields. To be as inclusive as possible, many of the terms found herein are not located in the manual, but are provided in order to supply definitions to common engineering acronyms and words.

DEFINITIONS

A

Abel — Advanced Boolean Expression Language. An early hardware description language developed for PLD-based designs.

Accelerator — A specialized piece of hardware that speeds up a software-based task. Usually used for speeding up simulation.

Algorithm — A recipe for performing an operation such as computing an average value.

Analog — A continuously varying signal. For example, if an analog signal's range is from 0 to 5 volts, the signal can assume any voltage within that range such as 1.2, 2.4 or 4.7 volts. Microprocessors and microcontrollers usually cannot process analog signals directly and require conversion by an A/D converter before they can process the signal.

Analog Simulation — Modeling or simulation of an electronic circuit using representations of the actual circuit voltages, currents, and component values instead of simplified digital state representations.

Analog-to-Digital Converter — A/D, ADC. An electronic circuit that converts a continuously varying signal (temperature, pressure, voltage, etc.) into digital zeroes and ones that can be processed by a microprocessor or microcontroller.

Analyzer, Logic — An instrument that allows you to observe the behavior of digital signals in an embedded system.

ASIC Application-Specific Integrated Circuit. A custom integrated circuit designed specifically for one end product or a closely related family of end products.

ASIC Emulation Also Logic Emulation. The use of programmable circuits usually based on FPGAs, to emulate the design of an ASIC or an IC before it is built. ASIC emulation allows designers to check the operation of a design before committing the time and money required to fabricate the IC. Emulation serves the same purpose as simulation design verification but is much faster because it is based on hardware rather than software.

Autohandler A machine that removes ICs from their shipping tubes, connects them to the programmer to be programmed or tested, and places them back in tubes. It may also have a marker that will label the devices. The most common autohandlers are manufactured by Exatron, MCT, and Quality Automation.

B

BGA Ball Grid Array. A surface mount device with solder balls and a high pin count, similar to PGA.

BiCMOS Bipolar, Complementary-Symmetry Metal Oxide Semiconductor. An integrated circuit fabrication technology that combines the two major IC technologies, bipolar and CMOS, on one IC.

Bidirectional A signal or port that can act as either an input to or an output from an electronic circuit.

Binary The base-2 number system almost universally used by modern computers, microprocessors, and microcontrollers.

Bipolar The original semiconductor manufacturing process technology. Usually characterized by high-speed, high-power operation.

Bipolar PROM A fuse-link programmable PROM.

Bit Contraction of Binary digiT. One digit in the base-2 numbering system used by virtually all modern computers, microprocessors, and microcontrollers. A bit can have a value of either zero or one.

Blank Check A test performed by a device programmer to ascertain whether a device has been programmed (partial or total) or is in a virgin state.

Block Diagram A graphical representation of a system using a very high level of abstraction.

Board, Circuit Also PC Board. A thin card, usually made from fiberglass or plastic, which is covered with copper lines and is used to hold the various

which is covered with copper lines and is used to hold the various integrated circuits in an embedded system.

Bond-Out Chip A special version of a microprocessor or microcontroller which brings critical internal signals from inside the chip out on special package pins so that developers can more easily observe what's happening inside the processor. Usually used to build In-Circuit Emulators (ICEs).

Boolean Algebra A mathematical system developed in the 1800s to express the philosophical logic of Aristotle, which was coincidentally ideal for the description of digital circuits 100 years later.

Breadboard A hand-made system prototype built as a proof of concept. In the early days of electronics (even before transistors were invented), engineers actually mounted circuit components to blocks of wood; hence the term "breadboard".

Buffer

1. An isolation circuit used to insulate sensitive analog or digital circuits from higher-power or higher-current levels in other portions of an electronic design. Often seen, for example, as an I/O buffer that separates the sensitive circuits inside of an IC from the signals on the circuit board to which the IC is attached.

2. Data storage unit directly stored on CPU.

Bus A group of two or more signals that carry closely associated signals in an electronic design.

Byte A binary word consisting of eight bits. When used to store a numeric value, a byte can represent a number from 0 to 255.

C

CAD Computer Aided Design. The overarching generic term for all software tools that enable or aid in the creation of engineered systems. Sometimes, CAD refers only to the electronic versions of mechanical drafting tools. Sometimes, it refers to all such tools including EDA tools.

CAE Computer Aided Engineering. The original term for electronic design automation (EDA). Now, often refers to the software tools used to develop the manufacturing tooling for the production of electronic systems such as for the panelization of circuit boards.

CFI Complex-Instruction-Set Computer. A design approach for microprocessors and microcontrollers that employs relatively complex instructions that execute over multiple clock cycles. A program written using CISC instructions requires fewer such instructions to perform a

task as compared to a program written using RISC (Reduced-Instruction-Set Computer) instructions.

Checksum A number that results by adding up every element of a pattern. Typically either a four or eight digit hex number, it is a quick way to identify a pattern, since it is very unlikely that two patterns will have the same checksum.

Clock A master timing signal that sets the operating pace for all other components in the embedded system.

Clock Skew Variation from the ideal clock timing across an entire electronic design (usually in an IC) caused by parasitic elements. Seymour Cray was an early combatant of clock skew and had to design serpentine traces on the Cray I supercomputer's circuit boards to compensate for clock skew.

Clock Tree A tree-like configuration of circuitry designed to minimize the effects of clock skew.

CMOS Complementary Symmetry Metal Oxide Semiconductor. An IC process technology developed in the 1960s that typically runs at lower power than bipolar circuitry. Early on, CMOS was much slower than bipolar but has steadily gained in speed over the decades to rival today's bipolar speeds. Most ICs are now made using CMOS technology.

Co-design See Hardware/Software Co-design.

Compare Reading a programmable device and displaying any discrepancies from the desired pattern. Each error is displayed on the screen. This comparison is slower to perform than a verify on the programmer.

Compiler A computer program which translates programs written in a high-level language (HLL) into assembly-language instructions or machine code.

Complex PLD These devices are the big brothers of PALs. Their architecture grew out of the familiar sum-of-products design that is common to PALs, but they may have as many as 84 pins, or more, buried logic that is not connected to the outputs, and more complex interconnection schemes.

Concurrency The ability of an electronic circuit to do several (or at least two) different things at the same time. Contrast with computer programs, which usually execute only one instruction at a time unless the program is running on a processor with multiple concurrent execution units.

Concurrent Design The ability to develop many parts of a complex electronic design in tandem using EDA tools such as simulation to stand in for portions of the system yet to be designed fully.

Concurrent Programmer — A multiple-socket programmer that starts programming each device as soon as it is inserted in a socket, without all sockets having to be filled.

Controller — An electronic system that directs the operation of some larger system.

Core — A predesigned block of logic employed as a building block for ASIC design.

Co-Simulation — Simulation of hardware and software together, simultaneously.

Coverage — A measure of the 'goodness' of a test or test suite. Usually refers to fault coverage and is expressed as a percentage of the circuit covered by the test. Usually, it is too expensive to achieve 100% coverage and test engineers shoot for coverage in the high 90's. Scan-test technology can improve coverage results, at the expense of additional silicon on the chip and some additional design time.

CPLD — Complex Programmable Logic Device. A programmable IC that is more complex than the original Programmable Logic Devices such as AMD's (originally MMI's) PALs but somewhat less complex than Field Programmable Logic Arrays.

CPU — Central Processing Unit. The core circuitry of a computer including the ALU (arithmetic logic unit), address-control circuitry, and bus-control circuitry. Usually implemented with a microprocessor or microcontroller in an embedded system.

Crosstalk — A condition where signal activity on one wire in an electronic circuit couples to another wire and causes noise through electrostatic (capacitive) or electromagnetic (inductive) coupling.

CUPL — A hardware description language originally developed for PLDs.

Cyberoptic — Computer generated; connection for light-pulse information to be transmitted

D

Debugging — The art of finding and eliminating errors in system designs.

Design Capture — Design Entry. The process of entering an electronic system design into a computer using EDA tools.

Design Error — A flaw designed into an electronic circuit, which is then faithfully reproduced in every manufactured system (as opposed to a manufacturing error that is a flaw created by the manufacturing process itself). Emulation, simulation, and design-rule-checking tools all help to minimize or eliminate design errors.

Design Rule Check Verification of an IC or PC board layout for conformance to the physical or electrical limitations of the implementation technology in use.

Device Microchip or Integrated Circuit chip.

DFT Design for Test. A design methodology that includes special attention to the design of a circuit and the addition of special circuitry that eases the testing of that design.

Dialog Box The method used by the device programmer's user-interface software to allow the user to select options and specify information. The user can specify any options and fill any blanks in the box then press ENTER to force the software to process the information.

Die The silicon chip that is located within an IC package. It is a small rectangular flat piece of silicon that has been fabricated with many transistors to perform a specific function. It is glued into a plastic or ceramic package and connected to the external metal interconnect pins of the IC with very small bonding wires. It can be seen through the window of erasable EPROMs.

Digital An approach to circuit design based on the binary number system. Signals in digital circuitry can only assume well-defined levels; intermediate levels are invalid. For example, in a digital system with a signal range of 0 to 5 volts, the digital signal may have the logical value of 0 if the signal voltage is within the range of 0 to 0.5 volts and a logical value of 1 if the signal voltage is within the range of 2 to 5 volts. Signal voltages of between 0.5 and 2 volts are invalid and are not allowed.

Digital Simulation A computer simulation of an electronic circuit that uses simple Boolean or logic states to represent the instantaneous state of the circuit. Because the representation is simplified from the actual voltage and currents present in the circuit, digital simulation is much faster than analog simulation.

Digital-to-Analog Converter D/A, DAC. A circuit that translates a signal from a numeric, digital representation used by microprocessors and microcontrollers into an analog signal.

DIP Dual Inline Package. An IC package with two rows of through-hole pins, usually on 0.1 pitch, 0.3 or 0.6 inches apart.

Documentation All of the paper and electronic documents supplied with a component or system that are absolutely critical to fully utilizing the product. For embedded systems designers and developers, there is never enough documentation.

DRC See Design Rule Check.

DSP Digital Signal Processor or Digital Signal Processing. A specialized microprocessor or electronic system designed to be very fast at processing continuous signals such as sound and video.

E

EDA Electronic Design Automation. A large collection of software tools that enhance and aid in the development of complex electronic systems.

EDIF Electronic Design Interchange Format. A standard representation format for describing electronic circuits used to allow the interchange of circuit design information between EDA tools.

EEPROM Electrically Erasable Programmable Read Only Memory. An integrated circuit that stores programs and data in many embedded systems. EEPROM stores retains information even when the power is off. Early EEPROM was expensive on a cost-per-bit basis and was infrequently used. Newer "Flash" EEPROM is much less expensive and its cost-per-bit approaches that of DRAM making Flash EEPROM a very attractive memory device for embedded-systems design.

EMI Electromagnetic Interference. Noise generated by electronic systems, which can interfere with other electronic systems by traveling through the air, over communication wires, and through power wiring.

Emulator ICE or In-Circuit Emulator. A complex, expensive, and often balky electronic system that simulates the presence of an embedded system's microprocessor or microcontroller. Used often and extensively in the development and debugging of embedded-system programs.

EPROM Erasable Programmable Read Only Memory/UVEPROM. An integrated circuit that stores programs and data in many embedded systems. EPROM can only be programmed once. To erase an EPROM's contents, it must be exposed to intense ultraviolet light for an extended length of time.

ESDA Electronic System Design Automation. High-level EDA tools used to design and describe entire electronic systems.

Event A point in time where a change occurs in the state of an electronic circuit.

Event-Driven Simulator A simulator that only calculates circuit conditions when events (changes) such as the start of a new clock cycle occur in the state of the system. In contrast, timing simulation computes the state of a system

using elapsed time (usually in nanoseconds or picoseconds).

F

Fast Prototype — A working product model built quickly to try out product concepts. May lack the fit, finish, and complete capabilities of the planned final product while still giving users an idea of how the product will work. Often assembled specifically for a conference or other dog-and-pony show.

Fault — An actual problem in an electronic circuit that disables or degrades the performance of the circuit. Also, for EDA purposes, a point in the circuit where a potential flaw could damage the circuit's operation.

Fault Coverage — The percentage of potential faults identified and tested by a test program or suite of test programs. If the tests can uncover all potential faults, the fault coverage for those tests is 100%.

Fault Simulation — Simulation of the operation of an electronic circuit with the introduction of simulated manufacturing faults to determine the amount of fault coverage provided by a set of test vectors. These test vectors are then used to test the actual manufactured circuit so they must be able to identify a large percentage of the possible manufacturing faults.

Finite Element Modeling — A relatively complicated numerical method (computer algorithm) that can model complex electrical phenomena such as electromagnetic wavefront propagation.

Floorplanning — The task of determining where each major block of circuitry will go within an IC design.

FPGA — Field Programmable Gate Array. A very complex PLD. An integrated circuit containing a large number of logic cells or gates that can be programmably configured after the IC has been manufactured. Some FPGAs use fuses for this programming and others store the configuration in an on-chip EEPROM or RAM memory. Fuse-programmed parts cannot be reprogrammed so they can only be configured once. EEPROM-based FPGAs can be erased and reprogrammed so they can be configured many times. RAM-based FPGAs can be reconfigured quickly, even while the circuit is in operation. The FPGA usually has an architecture that comprises a large number of simple logic blocks, a number of input/output pads, and a method to make random connections between the elements.

Framework — A unifying graphical user interface, database format, and inter-tool communication scheme that allows a user a to combine EDA tools from various vendors to create a desired tool suite for the design of electronic systems.

Functional Test

A test that is performed following the programming of a PLD. The test operates the device in its normal operating mode by simulating the inputs and outputs that the part will experience in normal operation. To perform the test, the engineer must supply a set of test vectors that describe normal operation of the device so the device programmer can apply the specified stimulus and verify that the device is operating as designed. It is important to perform a functional test on PLDs because, in many cases, the PLD cannot be fully tested at the factory before programming so a defective PLD may program correctly but fail the functional test. A properly designed functional test will verify that the part meets the design specification, ensuring that the device, the compiler, the programmer, and the engineer have all performed their respective tasks correctly.

Fuse

A metal connection within a PLD or memory that may be melted during programming to break the circuit. These links typically carry input signals to logic gates. Burning all the fuses except those that are required in the desired circuit forms the desired circuit configuration. Since the fuses cannot be tested nondestructively, fuse-like programmable devices cannot be 100% tested at the factory and consequently expected programming yields are usually 98-99%.

G

GaAs

Gallium Arsenside. A high-speed IC process technology that does not use silicon. Instead, GaAs uses the semiconductor element Gallium doped with the impurity Arsenic. GaAs process technology currently products the fasted ICs possible but advanced CMOS processing has greatly reduced the speed gap in the past few years. GaAs is most often used today for very high frequently radio circuits such as the transmitter circuits in cellular phones.

GAL

Generic Array Logic. EEPROM based second generation PAL devices.

Gang Programmer

A multiple-socket programmer that requires each device to be placed in a socket before any can be programmed. See *Concurrent Programmer*.

Gate Array

A type of ASIC in which the transistors, gates and other active circuit elements are fixed on a wafer called a “master slice”. The customization for a particular application is done using the metal interconnection layers on the chip. Thus, the IC vendor can fabricate and stockpile master slices well in advance of a customer order and then finish the fabrication by adding the metal layer or layers when the order is received. Because of this style of fabrication, gate arrays are the easiest ASICs to design and offer the fastest turnaround time between order and shipment of the finished parts. In the extreme, Chip Express offers laser-programmed gate arrays with 24-hour turnaround time or less.

Gbyte Gigabyte. 1,073,741,824 bytes.

GDS II A photoplotting file format usually employed for integrated circuit mask plotting files. Originally developed by GE Calma, an early EDA vendor.

Gerber Photoplot Gerber file. A de-facto file format standard originally developed for Gerber Scientific for its line of photoplotters. Usually used for representing printed-circuit board designs.

Ground Bounce Noise signals coupled into the grounding network of an electronic system that cause a variety of operating problems in the circuit. A phenomenon that limits the testability of high-speed PLDs on some device programmers. The term refers to the voltage on the ground terminal of the PLD's die rising and falling when many outputs switch simultaneously. This voltage can induce extraneous clock signals that will make a device fail a functional test or reduce programming yield.

H

Hard Macro A relatively complex block of logic or "core" such as a multiplier or an entire microprocessor that has been completely pre-designed for use on a particular ASIC or FPGA technology. Generally, a hard macro cannot be edited except by the company that created it. In exchange for this relative lack of flexibility, hard macros usually provide better performance using a smaller amount of silicon when compared to a "soft macro" or "synthesizable core".

Hardware/Software Co-design The simultaneous development of product hardware and software. This design approach is more difficult than a serial design, which first develops the hardware and then the software that will run on the hardware but the benefit is a reduced time to market. To develop software before hardware is ready, software developers often create a behavioral model of the hardware that can run the software and thus prove its function.

HDL Hardware Description Language. A synthetic computer-based language used for the formal description of electronic circuits. An HDL can describe a circuit's operation, its design, and a set of tests to verify circuit operation through simulation. The two most popular digital HDLs are VHDL and Verilog. An analog HDL called AHDL is under development by many vendors. HDLs make it easier to develop very large designs through formal software engineering methods that define ways to divide a large team through formal software engineering methods that define ways to divide a large team project into smaller pieces that can be implemented by individual team members.

Hex file A human-readable ASCII file that represents any binary data. Each byte in the binary pattern is represented by two hex characters (0-9, A-F) so

that any of the 256 possible bytes, which include both control and unprintable characters, may be printed. The hex file may also contain address or checksum information. The pattern represented by the hex file may be represented by a binary file or any of the hex file formats – any file format may contain any pattern. The names of the hex file formats (Intel, Motorola, Tektronix, etc.) indicate who standardized its format and does not indicate anything about the pattern or the device the pattern is intended for.

Hierarchy A method for describing and modeling an electronic system using different abstraction levels. At the bottom level of the hierarchy is the actual physical layout of the design (a concrete level, not at all abstract). At the top of the hierarchy is a functional description of the system or a block diagram (a very high level of abstraction). Intermediate levels include the register-transfer level (RTL), the gate level, and the transistor level.

HLL High-Level Language. A relatively complex computer programming language that allows the programmer to work at a mathematically abstract level instead of the low, physical level of the microprocessor or microcontroller. For example, instead of dealing directly with registers and memory locations, the HLL programmer works with variables and arrays. Java, C, Pascal, Fortran, and BASIC are all examples of high-level languages.

I

I/O Input/Output. The wide range of circuits and sensors used to bring information into an embedded-system processor and to transport processed information back out of the processor. Serial and parallel ports, keyboard and keypad controllers, floppy and hard disk drives, and displays are all examples of I/O devices.

IBIS I/O Buffer Information Specification. A standard simulation format used to model the behavior of an integrated circuit's input/output (I/O) pins. Used in designing and simulating the operation of circuit buses.

IC Integrated Circuit. A silicon chip containing hundreds, thousands, or millions of circuit elements such as transistors, resistors, capacitors, and inductors. RAM, ROM, microprocessors, and microcontrollers are all examples of integrated circuits.

J

JEDEC Joint Electron Device Engineering Council (pronounced JED'eck). A group organized by the IEEE (Institute of Electrical and Electronics Engineers) that has defined a standard file format for PLDs.

JEDEC file A file conforming to a standard format that specifies the configuration and testing procedure for a PLD. The file is in a human-readable ASCII format and consists of fields that start with a letter and end with an asterisk. Fields specify the pattern to program into the part, whether to secure the device, a set of test vectors to perform a functional test, and checksums to verify the integrity of the file.

K

Kbyte Kilobyte. 1024 bytes.

L

LCC Leadless Chip Carrier. A square ceramic package that has no leads; instead it has metal areas that are surface-mount soldered to the target circuit. This package is usually used only for military and aerospace applications. Available up to 84 pins.

Linear See Analog.

Logic Digital circuitry, whether in an IC, an ASIC, a microprocessor, or a microcontroller.

Logic Emulation See ASIC Emulation.

Logic Synthesis See Synthesis.

M

Make file A file commonly used by software engineers in conjunction with a make utility program to automate the building of software projects. If the software project required a device to be programmed following a compile and link operation, the make file can start the device programmer and specify a macro file to perform the programming operation.

Master Site The initial location where the first transaction of a job is performed. This is the first site a part will be programmed in before the job is broadcasted to all other sites. Traditionally, the Master Site is Site 1, but it can be any site available. Reasons for changing the Master Site location include a bad site, convenience of location, etc.

Mbyte Megabyte. 1,048,576 bytes.

MCM Multi-Chip Module. A hybrid manufacturing technique that places several IC chips into a single package. MCMs are a way of "creating"

an integrated circuit using otherwise incompatible IC fabrication technologies (such as CMOS and GaAs). MCMs are alos a way of extending the reach of existing ASIC technologies, which may lack the ability to implement an entire system design on one chip.

Memory device A device that contains an array of storage locations. The device has a set of inputs, called addresses, which specify which location in the array is being accessed. A set of input/output pins produce the stored number (pattern) when the device is read, and accept a new value when the device is written or programmed. Additionally, there are one or more input pins that select the operating move (read, write, standby, etc.). Memory devices may be classified by whether they are volatile or non-volatile, and whether they may be erased. The memory's organization refers to its word width and the number of words in the device.

Microcontroller mC. A real "computer on a chip", incorporating a microprocessor, memory, and I/O circuits on one integrated circuit. A device that contains a central processing unit (CPU), memory, and I/O ports on a single IC. Microcontrollers that contain any form of non-volatile memory may be programmed on a device programmer. When connected to a power supply and external crystal, many of these devices form a complete microcomputer. In many embedded systems, the microcontroller may well be the only integrated circuit in the design.

Microprocessor mP. The original "processor on a chip" introduced by Intel in 1971. An integrated circuit that contains all of the processing components of a computer CPU including the ALU, program sequencer, and bus interface. Newer microprocessors also incorporate cache memory for increased processing speed. Comes in 4-, 8-, 16-, 32-, and 64-bit varieties. Usually requires other ICs to make up an embedded system.

Microprocessor Emulator A piece of equipment substituted in a circuit for the circuit's microprocessor. The emulator gives more control over the circuit's operation and eases debugging and troubleshooting efforts.

MIPS Millions of Instructions Per Second. A performance figure of merit (numeric score or rank) for microprocessors and microcontrollers.

Mixed-Mode Operation in both the digital and analog domains (usually refers to simulation as in "mixed-mode simulation").

Mixed-Signal An electronic circuit that has both analog and digital sections. Because many "real-world" systems have analog interfaces (for example, most temperature, pressure, sound and video sensors are analog), most electronic systems must accommodate analog signals. However, signal processing is now most efficiently performed by digital circuits. Therefore, almost all modern electronic systems are mixed-signal systems although individual ICs in such systems need not be mixed-

signal chips. Instead, a design can achieve mixed-signal operation by combining separate analog and digital ICs.

Mixed-Signal Simulation — A simulation that combines the abilities of an analog simulation and a digital simulation. Used to verify the operation of mixed-mode circuitry.

Moore's Law — An empirical law developed and later revised by Intel's Gordon Moore, which predicts that the IC industry is capable of doubling the number of transistors on a silicon chip every 18 months (originally every year) resulting in declining IC prices and increasing performance. Most design cycles in the electronics in declining IC prices and embedded-system development firmly rely on Moore's law.

Multitasking — A programming style that splits the overall job to be performed by the embedded system into a number of smaller tasks, which then execute on the system's processor in a time-shared fashion.

N

Nanosecond — One billionth of a second.

Net — 1- For ASICs, an individual signal path including all of its branches and extensions.

2- An abbreviation for the Internet.

Net Extraction — The identification and cataloging of all signal paths in a circuit. The combination of all nets and circuit elements (transistors, resistors, capacitors, ICs, etc.) of an electronic design completely describes an electronic circuit.

Net List — Netlist. A computer file (sometimes a printed listing) containing a list of the signals in an electronic design and all of the circuit elements (transistors, resistors, capacitors, ICs, etc.) connected to that signal in the design.

Network Simulation — Simulation of a communications network to determine if it has the desired communications capacity, noise insensitivity, and fault tolerance.

Node — A single point in an electronic circuit.

Non-volatile — The characteristic of a memory that does not lose its contents when its power is removed. Non-volatile memory is useful in microcomputer circuits because it can provide instructions for a CPU as soon as the power is applied, before secondary devices, such as disk, can be accessed. Non-volatile memory includes ROM, EPROM and EEPROM.

O

Object-Oriented Programming — A programming styles that combines data blocks and the associated software processing algorithms into "encapsulated" modules with narrowly defined entry and exit points. This programming style was developed as a way of dealing with extremely large and complex software programming projects by breaking the project down into smaller chunks more easily handled by an individual programmer. The narrowly defined entry and exit points of each module prevent one programmer's module from disrupting another's.

Oscillator — A device that produces an alternating output current.

OTP — One-time programmable. The characteristic of a memory device that can be programmed once but cannot be erased. When an EPROM is described as OTP, this means that its die is erasable when exposed to ultraviolet light, but because of its package, which is not transparent, it cannot be exposed to light and thus it cannot be erased.

P

Package — The plastic or ceramic that protects an IC die and connects it to the target circuit.

PAL — Programmable Array Logic. The first truly successful family of programmable logic, originally introduced by Monolithic Memories in the early 1980s.

PALASM — PAL Assembler. The HDL originally developed by John Birkner of Monolithic Memories for the creation of PAL-based designs.

Parallel Printer Port — A standard port on virtually every PC designed for connection to a printer. This port has eight data lines and several control lines. Parallel ports may be either unidirectional or bi-directional. If your computer has a unidirectional port, the programmer will use the status lines to read data back from the programmer. The port allows high-speed communication (many times faster than a serial port). There may be up to three parallel ports in most PCs designated LPT1, LPT2, and LPT3.

Patch — A small piece of code used to repair an error in an existing embedded system program.

PCB — Printed circuit board, PC board, also PWB or Printed Wiring Board. A laminated board made from alternating layers of copper and plastic (usually impregnated with glass fibers for strength). The PC board serves as the physical carrier for other electronic components in an electronic design and also provides the electrical connection between

these electronic components.

PGA Pin Grid Array. A square, through-hold IC package that has pins located on a square grid with 0.1000-inch pitch. It may have up to several hundred pins. Used primarily for military and prototype designs.

Picosecond One trillionth of a second.

Place and Route A layout task that positions major functional blocks or electronic components within an IC or on a PC board (Place) and the subsequent routing of appropriate electrical connections to those components (Route).

Platform Term for a computer, operating system, or framework.

PLCC Plastic Leaded Chip Carrier. A low-cost square plastic package that has J-shaped leads on four sides. This can be surface-mounted or placed in a socket for through-hole use. PLCCs have interconnection leads on either two (usually only for memory chips) or all four sides (for logic and ASIC chips). Available in 20 to 84 pins.

PLD Programmable Logic Device. The generic term for all programmable-logic ICs including PLAs (programmable logic arrays), PALs, CPLDs (complex PLDs), and FPGAs (field programmable gate arrays).

PLD Compiler A software package that allows an engineer to specify the functionality of a PLD through a high-level language or schematic diagram. The software will convert the design into a JEDEC or other file for the PLD programmer. PLD compilers are available from numerous IC manufacturers and from third parties. The packages from IC manufacturers support only one brand of device and may be free, inexpensive or expensive. The most popular compiler is PALASM (priced under $200, available from AMD sales offices and representatives) which supports most of AMD's line of PMDs with an easy-to-learn high-level language. The compiler that probably offers the highest level of functionality and flexibility is PLDesigner made by MINC. It supports most PLDs and offers a sophisticated input language with full support for state machines and other complex constructs, partitioning designs into several PLDs, and graphical input. Their tools run on PCs and workstations. PLD compilers have simulators that can be used to test the functionality of your design and validate test vectors that you design before programming a device.

Pneumatic Of or pertaining to air, gases or wind.

Point Tool Term for an EDA tool that performs only one function.

Power Simulation A simulation that determines the power consumption of an electronic circuit operating under a variety of normal and abnormal conditions.

circuit operating under a variety of normal and abnormal conditions.

PQFP Plastic Quad Flat Pack. See *QFP.*

Price Point Term for price. Adapted from the consumer markets where there really is a difference in sales between products prices at $9.99 and $10.

PROM Programmable Read Only Memory. An integrated circuit that stores programs and data in many embedded systems. PROM stores/retains information even when the power is off but it can only be programmed or initialized once.

Q

QFP Quad Flat Pack. A square IC package that has surface-mount leads coming from four sides. It is used for high-density applications, usually over 100 pins. Lead pitch may be 0.025 inches or smaller.

R

RAM Random Access Memory. A volatile memory device. An integrated circuit that stores programs and data in many embedded systems. RAM does not retain information when the power is off and must therefore be reinitialized every time the embedded system is switched on. There are man varieties of RAM including the two most popular types: Dynamic RAM (DRAM) and Static RAM (SRAM).

RC Extraction The mathematical computation of an electronic circuit's fundamental circuit elements: resistors (abbreviated R), and capacitors (abbreviated C). RC extraction allows a simulator to determine the expected behavior of the electronic circuit through the mathematical modeling of simple circuit elements.

Register A location inside of a microprocessor, microcontroller, or I/O controller chip that stores control or status information.

RISC Reduced-Instruction-Set Computer. A design approach for microprocessors and microcontrollers, originally developed at IBM, which employs relatively simple instructions that usually execute in one clock cycle. This approach results in a faster, simpler processor design that uses fewer transistors. However, a program written using RISC instructions requires more instructions to perform a task as compared to a program written using CISC (Complex-Instruction-Set Computer) instructions.

ROM Read Only Memory. An integrated circuit that stores programs and data in many embedded systems. A non-volatile memory device that cannot be programmed by the user. It is programmed at the factory through the

use of a mask pattern in the final fabrication steps of the die. PROM stores/retains information even when the power is off but it can only be programmed or initialized once and only at the semiconductor factory.

ROM Emulator An embedded-system development tool that substitutes RAM for program ROM and aids in the debugging of the program.

RTL Register Transfer level or Register Transfer Logic. A register-level description of a digital electronic circuit (see Hierarchy). Registers store intermediate information between clock cycles in a digital circuit, so an RTL description describes what intermediate information is stored, where it is stored within the design, and how that information moves through the design as it operates.

S

Scan A specialized test approach that places special shift-register circuits inside of an electronic design just for test purposes. The shift register allows automatic test equipment to introduce test patterns deep into the circuitry and to read out status information that results from the circuit's response to those test patterns.

Schematic A graphical representation of an electronic circuit. Until the 1980s, schematics were really the only representation system used to describe circuits. However, with the advent of HDLs and an explosion in circuit complexity, schematics are becoming less important as a representation tool.

Schematic Entry The process of drawing a schematic using EDA tools. When done with paper and pencil, schematic entry is called schematic drafting or schematic drawing.

SCSI Small Computer System Interface. Pronounced "scuzzy". An 8-bit parallel computer peripheral interface standard used to connect to a wide variety of peripherals devices including hard disk and CD-ROM drives, tape-backup units, and optical scanners.

Sequencer A device for automatic or regulation of a sequence.

Serial Memory An EPROM or EEPROM that is accessed by shifting in addresses and shifting out data one bit at a time. Interfaces are available using one, two or three wires for clock, data in, and data out.

Simulation Modeling of an electronic circuit (or any other physical system) using computer-based algorithms and programming. Simulations can model designs at many levels of abstraction (system, gate, transistor, etc.). Simulation allows engineers to test designs without actually building them and thus can help speed the development of complex electronic

systems. However, the simulations are only as good as the mathematical models used to describe the systems; inaccurate models lead to inaccurate simulations. Therefore, accurate component models are essential for accurate simulations.

Simulation Model A software representation of a system component that describes how that component operates under various electrical and physical (temperature, pressure, light, etc.) stimulus.

Socket module An interchangeable metal chassis that contains a programming socket.

Soft Macro A predefined block of logic (such as a multiplier or microprocessor), which can be used as a building block for creating ASIC designs. In contrast to "Hard Macros", soft macros can be decomposed into component-level parts and edited for a particular application.

SOIC Small Outline Integrated Circuit. A surface-mount IC package that has two rows of leads on opposite sides. Commonly found in 8 to 32 pin sizes. Leads are usually 0.050 pitch.

SPICE Simulation Program with Integrated Circuit Emphasis. The original analog simulation program developed at the University of California Berkeley in the early 1970s.

SRAM Static Random Access Memory. An integrated circuit that stores programs and data in many embedded systems. SRAM does not retain information when the power is off and must therefore be reinitialized every time the embedded system is switched on. SRAM is more expensive than DRAM on a cost-per-bit basis but is usually easier to connect to a microprocessor or microcontroller.

Standard Cell A form of ASIC design that employs predefined logic cells and circuit components to create an ASIC. All mask layers of a standard-cell ASIC are custom for that ASIC, in contrast to a "Gate Array" in which only the metal-layer masks are custom. Standard-cell ASICs usually run faster and use less silicon (and are therefore usually cheaper on a per-part basis) than Gate Arrays. However, because the standard-cell ASIC uses predefined circuit components, its usually easier to design (and therefore requires less time to design) than a full-custom ASIC where every resistor, capacitor, and transistor is custom built.

State Diagram A graphical representation of a state machine's operation. State-diagram editors are EDA tools specifically designed to aid in the development of state machine designs.

State Editor A design-entry EDA tool used to create state diagrams.

State Machine A digital circuit built from registers and gates that controls the operation of other circuitry. For example, microprocessors contain many state

of other circuitry. For example, microprocessors contain many state machines that sequence the flow of information over the processor's bus and through its data-manipulation circuits.

Static Timing Analyzer An EDA tool that exhaustively checks every signal path in a circuit to identify timing-related design problems.

Symbol A graphic, schematic library element that represents an electronic component such as a resistor, a capacitor, a transistor, or an IC.

Symbol Editor An EDA tool for maintaining and creating schematic symbols.

Synchronous A digital circuit where all of the operations occur in lock step to a master clock signal.

Synthesis Logic Synthesis. A computer process that transforms a circuit description from one level of abstraction to a lower level, usually towards some physical implementation. Synthesis is to hardware design what compilation is to software development. In fact, logic synthesis was originally called hardware compilation.

T

Test Synthesis The automatic creation of test patterns and a test program for the verification of manufactured ICs.

Test Vector A stimulus pattern applied to a circuit to verify the circuit's operation. A set of characters that describe the inputs and outputs of a device during a functional test. There is one character in the vector for each pin on the device. Numbers represent inputs to be applied to the device (1 for Vih, 0 for Vil). Letters represent the outputs that must be tested (H for Voh, L for Vol, Z for high-impedance). During the test, the part will be powered up and each input will be applied to the device for the first vector. Then, each output will be applied to the device for the first vector. This process will continue for each vector and any errors will be reported.

Timing Diagram A graphical representation of the signals in an electronic circuit that shows how the signals change over time in relationship to each other.

Timing Simulation Simulation of an electronic circuit's operation over time using calculated circuit parameters such as resistance, capacitance, inductance, and timing delays.

Top-Down Design A design methodology that starts the design of an electronic system at the very highest level of abstraction and then methodically broadens the design through lower abstract layers until finally reaching the concrete, physical design layer which accurately represents the implementation

technology for the system.

TQFP Thin Quad Flat Pack. Similar to QFP but with a lower profile and physically smaller in length and width.

Transmission Line A conductor or wire that is suited to carrying high-frequency signals.

TSOP Thin Small Outline Package. A surface-mount package with fine-pitch leads (usually 0.025 inch pitch) on two sides. This package is very low profile and commonly available in a reverse (mirror image) pinout used to simplify circuit board layout. Usually 32 to 44 pins.

U

Unit Delay Simulation A simplified form of timing simulation where every digital gate is assumed to introduce one unit of delay to a signal. In reality, different gates have different speeds, but unit delay simulation trades off accuracy for simulation speed.

UV Erasable The characteristic of an EPROM that allows it to be erased with exposure to short-wave ultra-violet light. This high-energy light can discharge the floating-gate transistor cells that store bits in an EPROM. The most common source of such light is a mercury vapor tube much like an ordinary fluorescent tube, but without the phosphor that turns the UV light emitted by the mercury into visible light. The light from ordinary fluorescent lamps or sunlight generally takes years to erase an EPROM. All UV erasable parts have a quartz windowed ceramic package that allows exposure with UV light.

V

Verification The task of establishing the correctness of a design using EDA tools to automatically check the timing, connections, and rules used to design the circuit.

Verify Reading a programmable device and comparing its contents to the desired pattern for that device. This is a go/no-go test – it does not report what the discrepancies are. See also: *compare.*

Verilog A hardware description language developed by Gateway Design Automation (now part of Cadence) in the 1980s which became very popular with ASIC and IC designers.

VHDL VHSIC Hardware Description Language. A hardware description language developed in the 1980s by IBM, Texas Instruments, and Intermetrics under US government contract for the Department of Defense's VHSIC (Very High Speed Integrated Circuit) program. VHDL enjoys a growing popularity with ASIC designers as VHDL

development tools mature.

Volatile The characteristic of a memory device (specifically RAM) that will lose its contents when the power is removed. These parts are not programmable with a device programmer because they cannot be removed from the programming socket without losing their contents.

W

Word A unit of memory usually consisting of two bytes (16 bits).

Word width The number of output pins that a memory device has. The most common sizes for EPROMs is byte wide (8 bits) and "word" wide, or 16 bits. It can also refer to the aggregate width of several memory devices used in a set.

ACRONYMS

A

AAAI	American Association for Artificial Intelligence
AAL	ATM Adaptation Layer
ABET	Accreditation Board for Engineering and Technology
ACE	Advanced Computing Environment
ACL	Advanced CMOS Logic
ACM	Association for Computing Machinery
A/D and D/A	Analog-to-Digital and vice versa
ADI	Autodesk Driver Interface (from AutoCad)
ADPCM	Adaptive Differential Pulse-Code Modulation
AEA	American Electronics Association
AEW system	Airborne Early Warning System
AGC	Automatic Gain Control
AI	Artificial Intelligence
AIA	Aerospace Industries Association
AIAA	American Institute of Aeronautics and Astronautics
ALU	Arithmetic-Logic Unit
AMPS	Advanced Mobile Phone Service
ANSI	American National Standard Institute
API	Applications Programming Interfaces
AQL	Accepted Quality Level
Arpa	Advanced Research Projects Agency
ASCII	American Standard Code for Information Interchange
ASEE	American Society for Engineering Education
ASIC	Application-Specific IC
ASP	Average Selling Price (a sales term)
ASSP	Application-Specific Standard Product
ATA	AT Bus Attachment (a PC interface)
ATE	Automatic Test Equipment
ATM	Asynchronous Transfer Mode
ATPG	Automatic Test Pattern Generation
ATTC	The FCC's Advanced TV Test Center
AWACS	Airborne Warning and Control System

B

BCD	Binary-Coded Decimal
BiCMOS	Bipolar CMOS
BiFET	Bidirectional Field-Effect Transistor
BIOS	Basic Input/Output System
B-NTSC	Baseband National Television Standards Committee

C

CAD/CAM/CAE	Computer-Aided-Design, -Manufacture, -Engineering
CAGR	Compound Annual Growth Rate
CAS	Column-Address Strobe
CASE	Computer-Aided Software Engineering
CAU	Control Arithmetic Unit
CBI	Computer-Based Instrument
C^3	Command, Control and Communications Systems
CCD	Charge-Coupled Device
CCIR	Comite Consultatif International des Radio Communications
CCITT	Consultative Committee for International Telephone and Telegraph, former standards body of IEEE and now succeeded by the ITU-TSS, or International Telecommunications Union, Technical Standards Section
CD-I	Compact Disk-Interactive
CDPD	Cellular Digital Packet Data
CD-ROM	Compact Disk, Read-Only Memory
CEMOS	Complementary Enhanced MOS
CEPT	Conference of European Posts and Telecommunications
CERN	Conseil Europeen pour le Recherche Nucleaire aka the European Particle Physics Laboratory
CFC	Chlorofluorocarbon
CFI	CAD Framework Initiative
CFM	Cubic Feet per Minute
CGA	Color Graphics Adapter
Chill	CCITT High-Level Language
CHMOS	Complementary High-Performance Metal-Oxide Semiconductor
CIF	The CCITT's Common Intermediate Format (for video; 352-pixel x 288-line resolution)
CIM	Computer-Integrated Manufacturing
CIO	Counter/Timer Input/Output
CISC	Complex Instruction Set Computer
CMOS	Complementary Metal-Oxide Semiconductor
CMRR	Common Mode Rejection Ratio
COS	The Corporation for Open Systems, an organization of computer and communications equipment suppliers and users
COSE	Common Open System Environment
CPLD	Complex PLD
CP/M	Control Programs/Microcomputer
CPU	Central-Processing Unit
Crada	Cooperative Research and Development Agreement
CRC	Cyclic Redundancy Checking
CRT	Cathode-Ray Tube
CSA	Canadian Standard Association
CSIC	Customer-Specific IC
CSMA/CD	Carrier-Sense Multiple-Access with Collision-Detection (baseband schemes)
CTE	Coefficient of Thermal Expansion

CVD	Chemical Vapor Deposition
cw	Continuous wave

D

DAC	Design Automation Conference
DAT	Digital Audio Tape
DCC	Digital Compact Cassette
DCE	Data Circuit Terminating Equipment; also Distributed Computing Environment
DCFL	Direct-Coupled FET Logic
DCT	Discrete Cosine Transform
DDS	Digital Data Standard
DECT	Digital European Cordless Telephone (standard)
DESC	Defense Electronics Supply Center
DIP	Dual-In-Line Package
DSB	Display-Station Buffer
Divad	Initial cap; Division Air Defense
DMA	Direct Memory Access Controller
DMM	Digital Multimeter
DMSK	Differential Minimum Shift Keying
DOS	Disk Operating System, Operating System Standard created by Microsoft Corp.
DPMI	DOS Protected Mode Interface
dpst	Double-Pole, Single Throw; for switches or relays
DQPSK	Digital Quadrature Phase-Shift Keying
DRAM	Dynamic Random Access Memory
DRC	Design Rule Checking
DSO	Digital Storage Oscilloscope
DSP	Digital Signal Processing
DSU	Dataport Service Unit
DTE	Data Terminal Equipment
DTL	Diode Transistor Logic
DTMF	Dual-Tone Multifrequency
DUT	Device Under Test
DVI	Digital Video Interactive Technology

E

EAROM	Electrically Alterable ROM
ECC	Error-Correction Code
ECL	Emitter-Coupled Logic
ECO	Engineering Change Order
ECP/EPP	Enhanced Capabilities Port/Enhanced Parallel Port
ECU	European Currency Unit
EDA	Electronic Design Automation
EDAC	Electronic Design Interchange Format
EDIF	Electronic Design Interchange Format

EFTA	European Free Trade Area
EGA	Enhanced Graphics Adapter
EIA	Electronic Industries Association
EIAJ	Electronic Industries Association - Japan
EIC	Equivalent IC
EISA	Extended Industry Standard Architecture
EMC	Electromagnetic Compatibility
EMI	Electromagnetic Interference
EPLD	Erasable Programmable Logic Device
EPOS	Electronic Point of Sale
EPROM	Erasable PROM
EEPROM	Electrically Erasable PROM
ERC	Electrical Rule Checking
ESA	European Space Agency
ESD	Electrostatic Discharge
ESDI	Enhanced Small Device Interface
Esprit	European Strategic Program for Research and Development in Information Technology
ESR	Equivalent Series Resistance

F

FAE	Field-Application Engineers
FCC	Federal Communications Commission
FDDI	Fiber Distributed Data Interface
FDM	Frequency-Division Multiplexing
FET	Field-Effect Transistor
FFT	Fast Fourier Transform
FIFO	First In, First Out
FIN	Flexible Interface Network
FOIRL	Fiber-Optic Inter-Repeater Link
FPGA	Field-Programmable Gate Array
FSM	Finite-State Machine
FSR	Full-Scale Reading

G

GaAs	Gallium Arsenide
G&A	General and Administrative (expenses)
GATT	General Agreement on Tariffs and Trade
GDI	Graphical Driver Interface
GPIB	General-Purpose Interface Bus
GPS	Global-Positioning Satellite
GRIN	Graded Index (lens)
GSM	Global System for Mobile Communications (standard)
GTL	Gunning Transceiver Logic
GUI	Graphical User Interface

H

HAL	Hardwired Array Logic Device, from MMI (trademark)
HBTs	Heterostructure Bipolar Transistors
HDL	Hardware Description Language
HEMTs	High Electron Mobility Transistors
HiPPI	High Performance Parallel Interface

I

IC	Integrated Circuit
I-C bus	Inter-IC bus
ICCAD	International Conference on Computer-Aided Design
ICE	Integrated Circuit Engineering
IDE	Integrated Device Electronics (interface)
IEE	Institution of Electrical Engineers
IEEE	The Institute of Electrical and Electronics Engineers
IECQ	International Electrotechnical Commission of Quality Control System
IFPI	International Federation of Phonogram and Videogram Producers
IGBT	Insulated-Gate Bipolar Transistor
IGT	Insulated-Gate Transistor
IofNewt	Integration of Numerical and Experimental Wind Tunnels
IPC	Institute for Interconnecting and Packaging Electronic Circuits
IPI	Intelligent Peripheral Interface
IR LED	Infrared Light Emitting Diode
ISA	Industry Standard Architecture
ISDN	Integrated Services Digital Network
ISHM	International Society for Hybrid Microelectronics
ISO/IEC	International Standards Organization/International Electrotechnical Commission
ISSCC	International Solid-State Circuits Conference
ISV	Independent Software Vendor
ITC	International Test Conference
ITO	Indium Tin Oxide
ITU-T	International Telecommunications Union, Technical Standards Section. In 95% of the cases since 12/93, it has replaced CCITT (see above) as standards-setting body of IEEE. This is true for modern standards, cabling standards, ISDN and ATM standards, etc.

J

JAN	Joint Army-Navy Military Standard
JEDEC	Joint Electronic Device Engineering Council
JEIDA	Japan Electronics Industry Development Association
JEMI	Joint Equipment Manufacturers Initiative
Jessi	Joint European Submicron Silicon Initiative
JETRO	Japan External Trade Relations Organization
JIT	Just-In-Time; A Manufacturing Philosophy

JPEG	Joint Photographic Experts Group
JTAG	Joint Test Action Group

L

LAN	Local-Area Network
LCA	Logic Cell Array, from Xilinx (trademark)
LCC	Leadless Chip Carrier
LCD	Liquid-Crystal Display
LED	Light-Emitting Diode
LPE	Liquid Phase Epitaxy
LSI	Large-Scale Integration
LSB	Least Significant Bit
LSSD	Level-Sensitive Scan Design

M

MAC	Medium Access Control
MAN	Metropolitan-Area Network
MAP	Manufacturing Automation Protocol
MAU	Mathematics Acceleration Unit OR Media Attachment Unit, Depending on Context
MBE	Molecular Beam Epitaxy
MCB	Molded Circuit Boards
MCC	(yes, we know there's a letter missing in this acronym, but it's correct) Microelectronics and Computer Technology Corp.
MCM	Multichip Module
MESFETs	Metal-Semiconductor Field-Effect Transistors
MFM	Multifrequency Modulation
MIDI	Musical Instrument Digital Interface
Mips	Million Instructions Per Second
MIS	Metal-Insulator Semiconductor
MISFETs	Metal-Insulator Semiconductor Field-Effect Transistors
MITI	Ministry of International Trade and Industry
MLC	Multilayer Ceramic Capacitor
MMIC	Monolithic Microwave Integrated Circuit (generic designation)
MMU	Memory Management Unit
MOCVD	Metal-Organic Chemical Vapor Deposition
Mops	Millions of Operations Per Second
MOSFET	Metal-Oxide Semiconductor Field-Effect Transistor
MOSIGT	Metal-Oxide Semiconductor Insulated Gate Transistor
MPEG	Moving Picture Experts Group
Mpps	Megapixels Per Second
MPT	Ministry of Posts and Telecommunications
MRP	Manufacturing Resource Planning
MSI	Medium-Scale Integration
MTBF	Mean Time Between Failure
Mtops	Millions of Theoretical Operations per Second

MVS	IBM's Proprietary Operating System
MWh	Megawatt-hour

N

NAPLPS	North American Presentation Level Protocol Syntax, the ANSI-videotex graphics standard
NIC	Network Interface Controller
NII	National Information Infrastructure – the information superhighway
NISO	National Information Standards Organization
NIST	National Institute of Standard and Technology
NRE	Non-recurring Engineering (as in 'charges')
NSPE	National Society of Professional Engineers
NTSC	National Television Standards Committee

O

OCR	Optical Character Recognition
OECD	Organization for Economic Cooperation and Development
OEM	Original Equipment Manufacturer
OFDM	Orthogonal Frequency Division Modulation
OLTP	On-line Transaction Processing
OOPS	Object-Oriented Programming Software
OOPSLA	Object-Oriented Programming Systems, Languages and Applications conference
OROM	Optical ROM
OS	Operating System
OSF	Open Software Foundation
OSI	Open Systems Interconnection
OS/2	Operating System from Microsoft, IBM
OSTP	Office of Science and Technology Policy
OTP	One Time Programmable

P

PACE	Professional Activities Council Committees for Engineers
PAL	Programmable Array Logic
PA-RISC	Precision Architecture – Reduced Instruction Set Computer
PBX	Public or Private Branch Exchange
PCI	Peripheral Component Interconnect (local bus)
PCMCIA	Personal Computer Memory Card International Association (PCMCIA cards provide a standardized format and interconnection method for the addition of various peripherals, increased memory storage or other functions)
PCN	Personal Communications Network
PDA	Personal Digital Assistant
PIA	Peripheral Interface Adapter
PIN	Positive-Intrinsic Negative (diode)

PIV	Peak Inverse Voltage
PLA	Programmable Logic Array
PLCC	Plastic Leadless Chip Carrier
PLD	Programmable Logic Device
PRML	Partial-Response, Most-Likelihood (encoding technique)
PRO	Precision RISC Organization (HP consortium)
Promis	Project Management Integrated System, from Strategic Software Planning Corp. (trademark)
PSK	Phase Shift Keying
PSMA	Power Sources Manufacturers' Association
PSRR	Power Supply Rejection Ratio
PTT	Post, Telegraph and Telephone Administrations (a generic term for public telegraph and telephones; much like RBOC is for the seven baby bells)
PVM	Poly-Vector Modulation
PWM	Pulse-Width Modulation

Q

QAM	Quadrature Amplitude Modulation
QFP	Quad Flat Pack
QIC	Quarter Inch Cartridge

R

RAID	Redundant Arrays of Independent Disks
RAMDAC	Random Access Memory Digital to Analog Converter
RAM/ROM	Random Access and Read Only Memories
RARP	Reverse Address Resolution Protocol
RAS	Row Address Strobe
RBOC	Regional Bell Operating Companies
RC time constant	Resistance-Capacitance
R-DAT	Rotary Digital Audio Tape
R&D	Research and Development
RF	Radio Frequency
RFI	Radio Frequency Interference
RFP	Request for Proposal
RGB	Red Green Blue
RISC	Reduced Instruction Set Computer
RTP	Rapid Thermal Processing

S

SAR	Segmentation and Re-assembly
SAW	Surface Acoustic Wave
SCI	Scalable Coherent Interface
SCR	Silicon Controlled Rectifier
SCSI	Small Computer Systems Interface
SDLC	Synchronous Data Link Control

Secam	Systeme Electronique Couleur Avec Memoire. French TV transmission standard.
SEMI	Semiconductor Equipment and Materials International
SIA	Semiconductor Industry Association
SID	Society for Information Display
SIMD	Single Instruction, Multiple Data
SLIC	Subscriber Line Interface Circuit
SLIP	Serial Line Interface Circuit
SMA	Surface Mount Assembly; also Subminiature Type A (connector)
SMD	Surface Mount(ed) Device
SMDS	Switched Multimegabit Data Services
SMI	System Management Interrupt
SMPTE	Society of Motion Picture and Television Engineers
SMT	Surface Mount Technology
SNA	Systems Network Architecture (IBM protocol)
SNMP	Simple Network Management Protocol
SOI	Silicon On Insulator
SOJ	Small-Outline, J-leaded package
SOP	Small-Outline Package
SOS	Silicon-On-Sapphire
SPA	Software Publishers Association
SPACE	The Satellite Television Industry Association (Society for Private and Commercial Earth Stations)
SPEC	Systems Performance Evaluation Cooperative
Spice	Simulation program with integrated-circuit emphasis
SPIE	Society of Photometric and Instrumentation Engineers
SQUID	Superconductive Quantum Interference Device (from IBM)
SQL	Structured Quary Language
SRAM	Static RAM
SRQ	Service Request; an IEEE 488 command
SSI	Small Scale Integration
SSR	Solid State Relay
SSOP	Shrink Small-Outline Package
SVID	AT&T's System V Interface Definition, a definitive guide for programmers writing applications for System V.

T

TCE	Thermal Coefficient of Expansion (spell out on first mention)
TCP/IP	Transmission Control Protocol and Internetwork Protocol
TDMA	Time Division Multiple Access
TFT	Thin-Film Transistor
TFTP	Trivial File Transfer Protocol
T/H	Track and Hold
TOP	Technical Office Protocol
TOPFET	Temperature and Overload Protected Field-Effect Transistor
TOW missiles	Tube-launched, Optically tracked, Wire-guided missiles
tpi	Tracks per inch

TQFP	Thin Quad Flat Pack
TRON	The Real-time Operating System Nuclues project
TSOP	Thin Small-Outline Package
TTL	Transistor-Transistor Logic
TVRO	Television Receive Only; a type of satellite dish

U

UART	Universal Asynchronous Receiver/Transistor
UHF	Ultra-High Frequency
UIM	Universal Interconnect Matrix
UL	Spell out on first mention – Underwriters Laboratories
Unix	Computer Operating System
UPC	Universal Peripheral Controller; alsions (controller)
UPS	Uninterruptible Power Supply
USARTs	Universal Synchronous/Asynchronous Receiver/Transmitters
USC	Universal Serial Communicattage-controlled oscillator
UV	Ultraviolet

V

VAR	Value Added Retailer or Reseller
VAN	Value Added Network
VBR	Variable Bit-Rate
VCO	Volo Universal Price Code (no periods after the letters)
VDE	Verband Deutsher Elektrono-Techniker. West Germany's Components Safety Agency
VESA	Video Electronics Standards Association
VFC	Voltage to Frequency Converter
VFET	Vertical Field-Effect Transistor
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuit (US Government Program)
VITA	VME International Trade Association
Vital	VHDL Initiative Toward ASIC Libraries
VL-bus	VESA Local Bus (standard), developed by the Video Electronics Standards Association
VLDS	Very Large Data Storage
VLSA	Very Large-Scale Arithmetic
VLSI	Very Large-Scale Integration
VMEbus	One word, note caps and lower case
VMS	DEC's proprietary operating system
VTR	Videotape Recorder; use VCR

W

WAN	Wide Area Network

X

XGA	Extended Graphics Array

Z

ZIF	Zero Insertion Force
ZIP	Zigzag In-Line Package

INDEX

A

B

C

D

E

F

G

H

I

J

T

U

V

W

X

APPENDIX A
LIMITED WARRANTY

BP Microsystems, Inc. warrants this product against defects in material or workmanship for a period of one year, as follows:

For a period of one year from date of purchase, BP Microsystems, Inc. will repair or replace any defective product at no charge. Whether the warranted product should be repaired or replaced is wholly within the discretion of BP Microsystems, Inc.

This warranty does not cover any damage due to accident, misuse, abuse or negligence.

You should retain your dated bill of sale as evidence of the date of purchase.

REPAIR OR REPLACEMENT AS PROVIDED UNDER THIS WARRANTY IS THE EXCLUSIVE REMEDY OF THE PURCHASER. IN NO EVENT SHALL BP MICROSYSTEMS, INC. BE LIABLE FOR ANY SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES, NOR FOR ANY DAMAGES RESULTING FROM USE, MISUSE OR MODIFICATION OF THIS PRODUCT. EXCEPT TO THE EXTENT PROHIBITED BY APPLICABLE LAW, ANY IMPLIED WARRANTY OR MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE IS SPECIFICALLY DISCLAIMED. UNDER NO CIRCUMSTANCES SHALL BP MICROSYSTEMS, INC. TOTAL LIABILITY EXCEED THE PURCHASE PRICE OF THE PRODUCT.

This warranty gives you specific legal rights, and you may also have other rights that vary from state to state.

APPENDIX B ADVANCED FEATURE SOFTWARE

This appendix covers information regarding the Advanced Feature Software (AFS).

AFS adds additional testing and production features, such as the autohandler interface to your BP software. AFS is only available for the BP-1200 Universal Device Programmer.

UPGRADING YOUR SOFTWARE

It is possible to upgrade the standard software on your BP-1200 to the AFS software.

To upgrade your existing software, first obtain an authorization code from BP Microsystems, Inc. You will need to have the version number of your software and your programmer serial number available when you request the upgrade code. You must now start the software and use the *AFS/Upgrade* command to enter your authorization code. The program will modify its own bp.exe file to enable the option. Verify that AFS has been enabled by looking at the bottom of the screen, which will indicate AFS on the Status Line.

☐ *If you ordered AFS when you ordered your programmer, or if you upgrade to a new version of the software, the AFS features will be enabled at the factory; you will not need an authorization code.*

AUTOHANDLER

The Autohandler mode interfaces between the operation of the handler and the programmer. The Autohandler selection is located in the *Device/Handler* command. We currently support several autohandler models by Exatron (both serial and parallel interfaces) and by MCT. Although the Handler menu always appears in the software, only the manual mode may be used without the purchase of AFS. Please refer to the description of the

Device/Handler command in the Command Reference chapter for more information on the configuration and use of this feature.

More information on using the BP-1200 with an Autohandler is being assembled. If you need more information, please contact BP Microsystems, Inc. and someone will assist you.

SERIALIZATION

The Serialize feature allows you to serialize devices using a starting number read from a specified file. A file will be created for you if the one you specify does not exist. The number will be incremented by 1 for each device operation and loaded into a specified buffer address prior to each operation.

The following options can be set using the *AFS/Serialize* command:

Serial number size: Specifies the length of the serial number a 1, 2, or 4 bytes.

Serial number data file: Specifies the name of the file containing the starting serial number. A file will be created with the specified name if one does not already exist. The file will be updated when the serial number is incremented.

Buffer address for serial number: Specifies the buffer address of the first byte of the serial number.

Buffer bytes to skip: Allows programming the serial number into only the even (low) or odd (high) byte addresses.

Byte order: Specifies REVERSE or FORWARD byte order. REVERSE is used by Intel processors and will place the LSB in the first byte of the serial number word. FORWARD is used by Motorola processors and will place the MSB in the first byte of the serial number word.

VERIFY CHECKSUM

The Verify Checksum feature automatically performs a checksum operation after programming or verifying a device and compares this sum to the reference checksum entered by the user. The checksum feature provides an independent means to confirm that the correct pattern has been programmed into the device. The feature is enabled by using the *AFS/Options* command.

There are two modes for Verify Checksum: manual and automatic modes. When manual mode is selected, the user will be prompted to enter the reference checksum. In this mode, this feature will detect file load errors, communication errors, hard disk errors, RAM errors, operator errors and configuration errors – all the while guaranteeing the highest level of data integrity. If the user inadvertently loads the wrong file, for instance, the sum computed by the checksum operation will not match the sum that the user entered, thus producing an error message. When the automatic mode is selected, the reference checksum is set automatically when a file is loaded; this mode is useful to detect RAM errors and some configuration errors.

REMOTE

The Remote feature allows the BP software to receive its commands and data over the serial port rather than from the keyboard. The commands are a subset of the commands available from other programmer manufacturers. Currently, Exatron sells software and autohandlers which use this feature of the software.

The only options for the Remote command are used to configure the serial port which is used to communicate with the remote host. The following options can be set using the AFS/Options command:

Remote port: Specifies the serial port to which your remote host is connected. The computer on the other end of this connection will provide commands to be executed here. Valid options are COM1, COM2, COM3, or COM4. The serial port selected must be controlled by a 16550 UART.

Remote baud rate: Specifies the baud rate to be used. The software will support any valid baud rate, but the use of 1200, 9600, or 19200 is suggested.

Remote data bits: Specifies the number of data bits to be used. Valid numbers are seven (7) and eight (8).

Remote parity: Specifies the parity to be used. Valid parity settings are NONE, ODD or EVEN.

Remote stop bits: Specifies the number of stop bits to be used. Valid numbers are one (1) and two (2).

The remote host must be configured with the same baud rate, number of data bits, parity setting, and number of stop bits as the serial port. Also note that the remote software uses the RTS/CTS and/or XON/XOFF flow control methods.

The AFS/Remote command is used to enter Remote mode. Once the Remote mode is entered, the software waits to receive a command over the serial port. When it receives a command, it will perform the action required. To exit the Remote mode, the remote host may send the Z command, or the ESC key may be pressed.

The commands used to communicate in Remote mode are one or three character strings. Each command is terminated with a carriage return. A command may be preceded by parameters. Each command generates a response back to the host, consisting of a return value, a response code, and a carriage return. The response code is '?' for commands which were not understood, 'F' for commands which failed, and '>' for commands which succeeded. The format of these commands is defined by the Data IO CRC commands.

The following table is a summary of the commands that are available:

CMD	Description
A	Enter translation format
:	Select device begin address
;	Select memory block size
<	Select memory begin address
W	Set I/O offset
03]	Set device ID verify option
22]	Set data word width
23]	Select number of verify passes
24]	Select security fuse programming option
26]	Specify logic verify options
27]	Set/clear enable/disable security fuse
2A]	Enable programming options
2B]	Disable programming options
2D]	Vector test options
R	Return status of device
33]	Select device manufacturer
34]	Select device part number
40]	Upload parts list
B	Blank check
L	Load RAM from device
P	Program device
T	Illegal-bit test
V	Verify device
F	Error status inquiry
X	Error code inquiry
01]	Display system configuration
46]	Clear yield tally
H	No operation
Z	Exit remote control
D	Set odd parity
E	Set even parity
J	Set 1 stop bit
K	Set 2 stop bits
N	Set no parity
04]	Set remote port baud rate
06]	Select data bits
^	Clear/fill RAM with data
=	Select I/O timeout
I	Input from port
M	Enter record size
O	Outport to port
S	View sumcheck
U	Set nulls
02]	Set upload wait time

2C]	Select memory fill option
2F]	Return 8-character sumcheck

FOR MORE INFORMATION

For more help and/or information while running the software, highlight a command and press <**F1**>.

APPENDIX C
BP-1200 UPGRADE PROCEDURES

SELF-TEST

Before beginning the Upgrade Procedures, you will need to run the self-diagnostic test on the programmer to determine whether or not you will need to replace the firmware (BIOS chip). Start the test by pressing <**ALT-D**>. The firmware version will be printed on the screen during the beginning of the test. If the current firmware version is less than V1.11, you will need to replace the BIOS chips. If it is V1.11 or greater, a replacement is not needed.

☐ *If the firmware needs to be replaced, contact BP Technical Support at 1-800-225-2102 to order new chips.*

BIOS REPLACEMENT

Opening the Base Unit

1. Open the Base Unit by removing four (4) screws – two (2) are located in the front and two (2) are located in the back.

2. Remove the top chassis and set it vertically next to the bottom, taking care not to disconnect any wires running between the two.

Replacing the Firmware

1. Remove the two (2) BIOS chips in socket locations U8 and U9 (see Figure B).

2. Insert the new chips, making sure to put U8 in socket location U8 and U9 in socket location U9.

UPGRADING

Once the firmware has been updated (or confirmed), you may proceed with upgrading procedures.

INSTALLING PIN DRIVER CARDS

Notice how the pin driver cards are facing and insert your new pin driver card(s) into the next available slot(s) facing the same direction.

- **Before closing the Base Unit, check to ensure that all cable connections are secure.**

CLOSING THE BASE UNIT

1. Close up the Base Unit by placing the top chassis on the bottom and taking care to route the cables around the pin driver cards and not over the top of the cards.

2. Secure the top and bottom chassis by replacing the four (4) screws taken out earlier

COMPLETING THE UPGRADE

- **When securing the Technology Adapter, use only a ½ turn on the screw. More than ½ a turn may fracture the PCB.**

1. Install the Technology Adapter. Place the Technology Adapter on top of base unit and secure the screw with a ½ turn.

☐ *The two (2) connectors are keyed so there is no possibility of installing it backwards.*

2. Replace the Socket Module. Again, the two (2) connectors on the Socket Module are keyed so it cannot be installed incorrectly.

3. Connect the power and parallel cables. Turn on the programmer and run the Self-Diagnostic test by pressing <**Alt-D**> again.

- **If the self-test fails or you encounter any other problems, please call our Technical Support staff at 1-800-225-2102.**

BP-1200 Modular Design

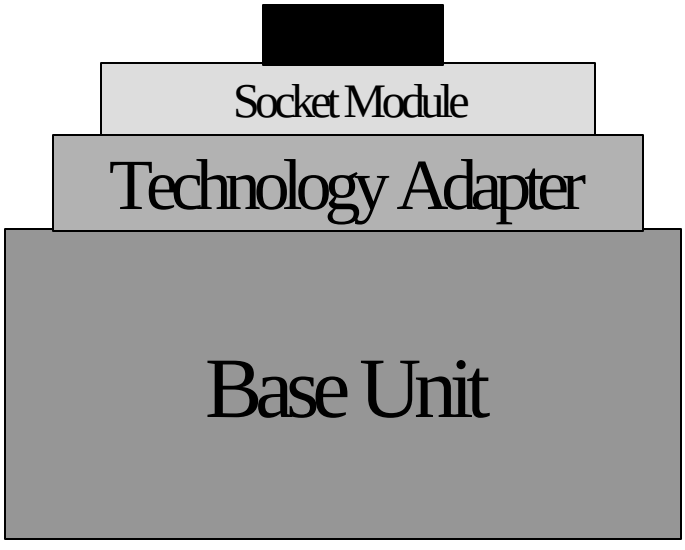


Figure A

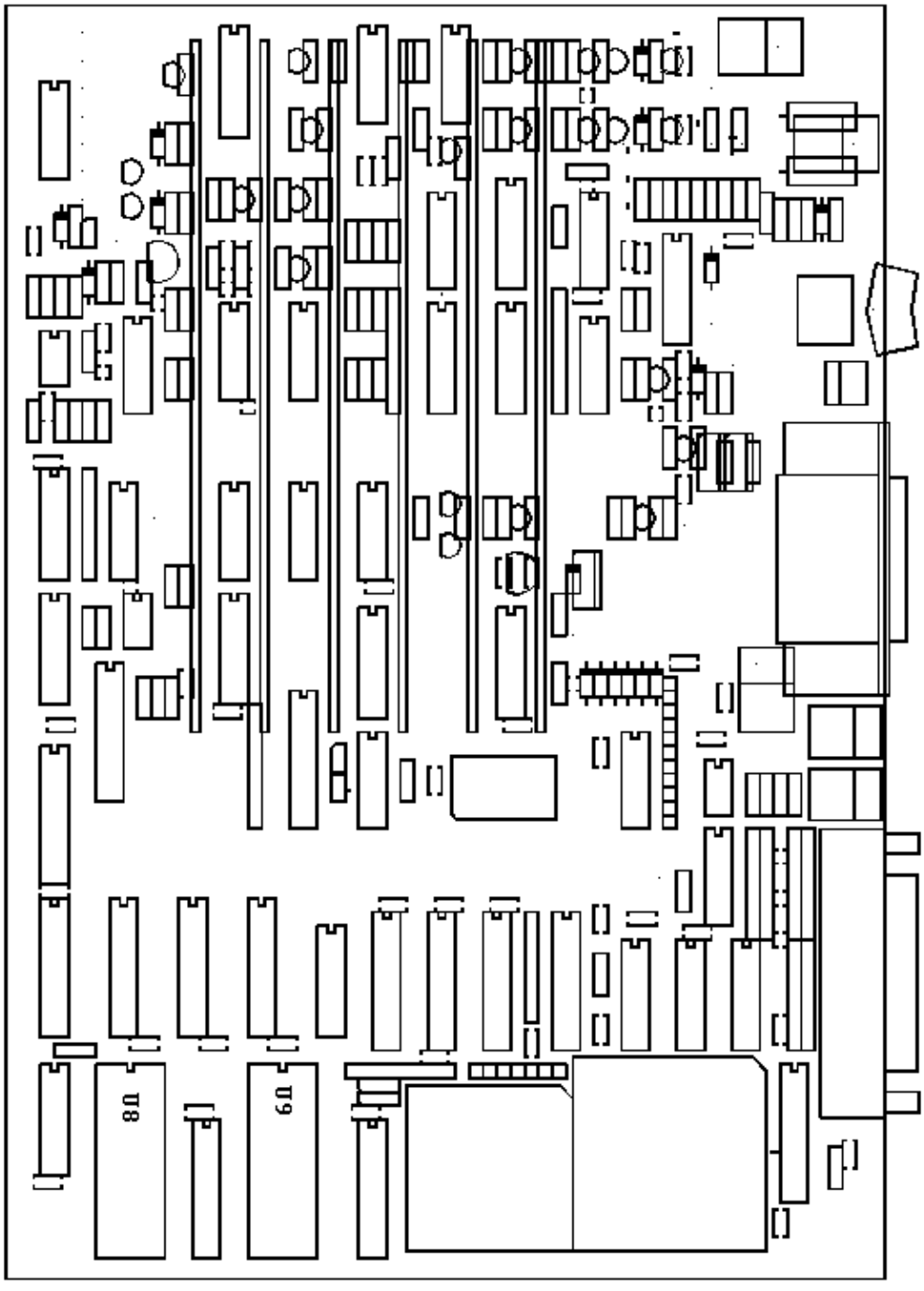


Figure B

APPENDIX D
QUICK START GUIDE

If you are an experienced user of BP Microsystems Programmers, this is a quick guide to getting your programmer up and running.

1. Connect all system components (see Chapter 2 – Getting Started, page 2-1).

2. Power up the programmer(s). Allow self test to complete.

3. Power up PC and monitor.

4. Put on ESD wrist-strap and plug into grounded receptacle.

5. Start BP software by typing <**BP**> at the DOS command prompt and pressing <**Enter**>.

6. Select Device to be programmed, <**Alt-S**> (see *Chapter 6 – BP Software Command Reference, page 6-1*).

7. Insert device in master programmer site.

8. Read master device, <**Alt-R**>, and remove or load file <**Alt-L**>.

9. Set Number of Devices (*Device/Handler* menu).

10. Program first device, <**Alt-P**>.

11. Continue to insert blank devices until done.

NEED HELP?

Remember, you can get context-sensitive help at any time by pressing the <**F1**> function key.

For more detailed operating instructions, see *Chapter 2 - Getting Started* and *Chapter 3 – Programming from Start to Finish.*

For information on specific commands, see *Chapter 6 – BP Software Command Reference.*

APPENDIX E
CHECKLIST

PACKAGE CHECKLIST

The programming system is delivered in at least two boxes. If the PC option is also purchased, two additional boxes containing the PC and its monitor are included. Please check each box carefully to insure the correct equipment has been received and is undamaged.

Package Summary

BP Programmer x2, x4, x6, or x8 with the SM48D socket modules
Certificate of Conformance
Registration Card
Additional socket modules (if applicable)
Power Cable
Communication Cable
Software Diskettes (BP.EXE)
System Manual

486 Class PC
Power Cable
PC Registration Card

PC Monitor