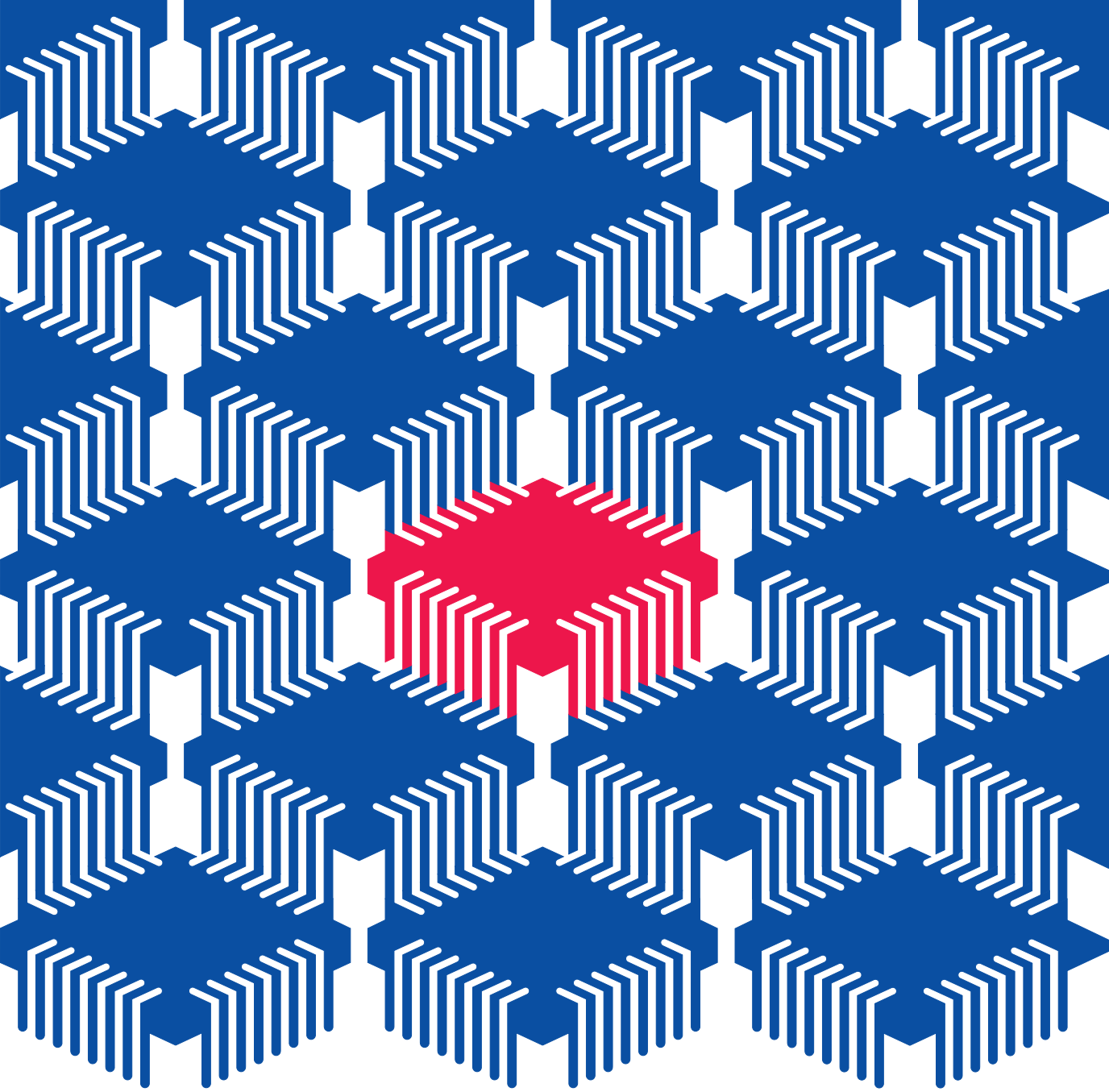


MAX+PLUS® II GETTING STARTED

MAX+PLUS® II

Programmable Logic Development System

Getting Started

Altera Corporation
101 Innovation Drive
San Jose, CA 95134
(408) 544-7000

MAX+PLUS II Getting Started
Version 8.1
September 1997 P25-04803-03

Printed on Recycled Paper

I.S. EN ISO 9001

Contents

Preface

Section 1 MAX+PLUS II Installation

Section 3 MAX+PLUS II Tutorial

Illustrations

Tables

MAX+PLUS II Fundamentals

This section describes the MAX+PLUS II manual and on-line help documentation and conventions. You should be familiar with this information before using MAX+PLUS II documentation.

MAX+PLUS II Documentation

MAX+PLUS® II documentation is designed for the novice as well as for the experienced user. It includes manuals and extensive, illustrated Help.

MAX+PLUS II Documents

MAX+PLUS II systems include the following documents:

MAX+PLUS II Getting Started	Contains step-by-step instructions on how to install MAX+PLUS II hardware, software, and licenses on PCs and UNIX workstations. It also provides an overview of the entire MAX+PLUS II system, and a tutorial that takes you from design entry to device programming. In addition, it contains information about MAX+PLUS II command-line operation and Altera's support services. Free electronic copies of this manual are also available from Altera's world-wide web site at **http://www.altera.com**.
MAX+PLUS II AHDL	Contains complete information on the Altera Hardware Description Language (AHDL™), including a detailed *How to Use AHDL* section with many examples.
MAX+PLUS II VHDL	Provides information on how to use the Very High Speed Integrated Circuit (VHSIC) Hardware Description Language (VHDL) with MAX+PLUS II, including a *How to Use MAX+PLUS II VHDL* section with many examples.
MAX+PLUS II Verilog HDL	Provides information on how to use the Verilog Hardware Description Language (HDL) with MAX+PLUS II, including a *How to Use MAX+PLUS II Verilog HDL* section with many examples.
MAX+PLUS II Help Poster	Provides handy and colorful descriptions of how to use on-line help in MAX+PLUS II.

MAX+PLUS II also includes the following Software Interface Guides, which are available in the **\lit** directory of the MAX+PLUS II CD-ROM as Adobe Portable Document Format (PDF) files. These guides provide detailed step-by-step examples and important guidelines:

- ***Cadence & MAX+PLUS II Software Interface Guide***
- ***Mentor Graphics & MAX+PLUS II Software Interface Guide***
- ***Synopsys & MAX+PLUS II Software Interface Guide***
- ***Viewlogic Powerview & MAX+PLUS II Software Interface Guide***

These Software Interface Guides are also available from Altera's world-wide web site at **http://www.altera.com**.

The MAX+PLUS II CD-ROM also includes the Adobe Acrobat Reader 3.0, which you can install on your hard drive in order to read the PDF files. Go to "Installing the Adobe Acrobat Reader" on page 51 for more information.

MAX+PLUS II Help

Your primary source of information on MAX+PLUS II is the complete on-line help. All of the information necessary to enter, compile, and verify a design and to program an Altera device is available in MAX+PLUS II Help.

Help also provides introductions to all MAX+PLUS II applications, guidelines for designing circuits with MAX+PLUS II, pin and logic cell numbers for each Altera device package, and summaries of other Altera documents, such as application notes, that can assist you with logic design.

How to Use MAX+PLUS II Documentation

How you use MAX+PLUS II documentation depends on your level of expertise and your approach to learning how to use a new tool.

If you are a novice user, you should take time to read the ***MAX+PLUS II Getting Started*** manual and complete the *MAX+PLUS II Tutorial* on page 155. Once you begin using MAX+PLUS II applications, you will find that the easy-to-use, extensive on-line help can quickly turn you into an expert MAX+PLUS II user. For basic information on using on-line help, refer to the *MAX+PLUS II Help Poster*. More detailed information on using Help is available in *MAX+PLUS II — A Perspective* on page 73.

If you are an experienced circuit designer or one who prefers to learn by experimenting, you will find the on-line help invaluable. Context-sensitive and menu-driven help give instant access to all MAX+PLUS II information.

Regardless of your level of expertise, you must follow the installation instructions provided in *MAX+PLUS II Installation* on page 1. Before you install the MAX+PLUS II hardware and software, you should also read the **read.me** file, located in the top-level directory of the MAX+PLUS II CD-ROM. Once you have installed MAX+PLUS II, you can open the **read.me** file through the Help menu in MAX+PLUS II.

If you are using EDA tools from Cadence, Mentor Graphics, Synopsys, or Viewlogic (Powerview), you should read the Software Interface Guide provided in the **\lit** directory of the MAX+PLUS II CD-ROM.

Altera Applications Engineers are also available to answer your questions. For more information about Altera's technical support services, see *Appendix A: Altera Support Services* on page 281.

Documentation Conventions

MAX+PLUS II manuals and MAX+PLUS II Help use the following conventions to make it easy for you to find and interpret information.

Terminology

The following terminology is used throughout MAX+PLUS II Help and manuals:

Term:	Meaning:
Button 1	Left mouse button.
Button 2	Right button on a two-button mouse, or middle and right buttons on a three-button mouse.
"point to"	Indicates that you should move the mouse so that the pointer is over the specified item.
"press"	Indicates that you must hold down a mouse button or key.
"click"	Indicates a quick press and release of a mouse button.
"double-click"	Indicates two clicks in rapid succession.
"choose"	Indicates that you need to use a mouse or key combination to start an action. For example, when you use the mouse to choose a button, you point to the button and click Button 1. When you use the keyboard to choose a command, you press Alt and then type letters that are underlined in the menu bar and menu.
"select"	Indicates that you need to highlight text and/or objects or an option in a dialog box with a key combination or the mouse. A selection does not start an action. For example: Select the AND2 primitive, then choose **Delete** from the Edit menu.
"turn on"/"turn off"	Indicates that you must click Button 1 on a checkbox or choose a menu command to turn a function on or off.

Typographic Conventions

MAX+PLUS II documentation uses the following typographic conventions:

Visual Cue:	Meaning:
Bold Initial Capitals	Command names, dialog box titles, and button names are shown in bold, with initial capital letters. For example: **Find Text** command, **Save As** dialog box, and **Start** button.
bold	Directory names, project names, disk drive names, filenames, filename extensions, and software utility names are shown in bold. Examples: **\maxplus2** directory, **d:** drive, **chiptrip.gdf** file. These items are not case-sensitive in the Windows environment; however, they are case-sensitive in the UNIX workstation environment. MAX+PLUS II Help shows these items in the case appropriate to the workstation environment.
Initial Capitals	Keyboard keys, user-editable application window fields, and menu names are shown with initial capital letters. For example: Delete key, the Start Time field, the Options menu.
"Subheading Title"	Subheadings within a manual section are enclosed in quotation marks. In manuals, titles of help topics are also shown in quotation marks.
Italic Initial Capitals	Help categories, section titles in manuals, application note and brief names, checkbox options, and options in dialog boxes are shown in italics with initial capital letters. For example: *Text Editor Procedures*, the *Check Outputs* option, the *Directories* box in the **Open** dialog box.
italics	Variables are enclosed in angle brackets (< >) and shown in italics. For example: <*filename*>, <*project name*>**.acf** file.
Bold Italics	Manual titles are shown in bold italics with initial capital letters. For example: ***MAX+PLUS II Getting Started***.

Visual Cue:	Meaning:
`Courier font`	Anything that must be typed exactly as it appears is shown in Courier. For example: `c:\max2work\tutorial\chiptrip.gdf`. Also, sections of an actual file, such as a Report File, references to parts of files (e.g., the AHDL keyword `SUBDESIGN`), and logic function names (e.g., `DFF` and `16cudslr`) are shown in Courier.
`Bold Courier font`	In syntax descriptions, bold Courier may be used to help distinguish literal text from variables.
1., 2., 3.,…, a., b., c.,…, *and* i., ii., iii.,…	Numbered steps are used in a list of items when the sequence of the items is important, such as the steps listed in a procedure.
■	Bullets are used in a list of items when the sequence of the items is not important.
v	The checkmark indicates a procedure that consists of one step only.
	The hand points to information that requires special attention.
	In MAX+PLUS II manuals, the feet show you where to go for more information on a particular topic.
	In MAX+PLUS II Help, the upward-pointing hand indicates that you can click Button 1 (the left mouse button) on any portion of the illustration that follows it to get help on that item. The mouse pointer changes to an upward-pointing hand when it is over a picture or word for which help is available.
Special symbols	Special symbols are used for these items: ↵ Enter key ⌐ Low-to-high transition ¬ High-to-low transition

Key Combinations

Key combinations and sequences appear in the following format:

Format Cue:	Meaning:
Key1+Key2	A plus (+) symbol indicates that you must hold down the first key when you press the second key. For example: Ctrl+L means that you must hold down Ctrl while pressing L, then release both keys.
Key1,Key2	A comma (,) indicates that you must press the keys sequentially. For example: Alt,F1 means that you must press the Alt key and release it, then press the F1 key and release it.

Backus-Naur Form

The Backus-Naur Form (BNF) defines the syntax of the text file formats and message variables. BNF uses the following notation:

Characters:	Meaning:
::=	"is defined as"
<...>	Identifiers (i.e., variables)
[...]	Optional items
{ ... }	Repeated items (zero or more times)
... | ...	Indicates a choice between items
:n:n	Suffix indicates a range (e.g., *<name char>:1:8* means "from 1 to 8 name characters")
italics	Variables in syntax descriptions
`Courier font`	Literal text in syntax descriptions. Bold Courier font is sometimes used to help distinguish literal text from *italic variables* in syntax descriptions.

MAX+PLUS II Help Updates

MAX+PLUS II Help is updated whenever the MAX+PLUS II software is updated; therefore, the on-line information is always current.

If you find a discrepancy between a MAX+PLUS II manual and the MAX+PLUS II on-line Help, you should rely on the MAX+PLUS II Help information.

You can get information on changes to MAX+PLUS II software and Help by choosing **New Features in this Release** (Help Menu) in MAX+PLUS II. Late-breaking news on Help and software is also available with the **READ.ME** command (Help menu).

Between MAX+PLUS II software releases, check the Altera world-wide web page for additional news and information, including help from the Atlas solutions database. Go to **http://www.altera.com**.

Sample Files

A variety of sample design files are copied to your hard disk when you install MAX+PLUS II. The installation procedure automatically creates subdirectories for these files.

The pathnames below are shown using the PC pathname convention of backslash (\) characters, but UNIX pathnames use forward slash (/) characters. On a UNIX workstation, the **/max2work** directory is a subdirectory of the **/usr** directory. Otherwise, the file and directory organization is identical.

- The **\max2work\chiptrip** directory contains all files for the **chiptrip** tutorial project that is described in ***MAX+PLUS II Getting Started***.
- The **\max2work\ahdl** directory contains all sample files used to illustrate AHDL features in MAX+PLUS II Help and in the ***MAX+PLUS II AHDL*** manual.
- The **\max2work\vhdl** directory contains all sample files used to illustrate VHDL features in MAX+PLUS II Help and in the ***MAX+PLUS II VHDL*** manual.
- The **\max2work\verilog** directory contains all sample files used to illustrate Verilog HDL features in MAX+PLUS II Help and in the ***MAX+PLUS II Verilog HDL*** manual.
- The **\max2work\edif** directory contains all sample files used to illustrate EDIF features in MAX+PLUS II Help.

Go to "MAX+PLUS II File Organization" on page 69 for more information about MAX+PLUS II directory structure.

Go to the Altera-provided Software Interface Guide for your third-party environment for information on the directory structure and sample files installed for third-party interfaces to MAX+PLUS II.

About *MAX+PLUS II Getting Started*

MAX+PLUS II Getting Started contains the following sections:

Section 1: MAX+PLUS II Installation gives hardware, software, and license installation instructions for PCs and UNIX workstations.

Section 2: MAX+PLUS II—A Perspective is an introduction to MAX+PLUS II software. It describes the on-line help and all MAX+PLUS II applications.

Section 3: MAX+PLUS II Tutorial takes you step-by-step through all facets of designing logic with MAX+PLUS II. It demonstrates three design entry methods and shows how to compile and simulate a project, analyze timing performance, and program an Altera device.

Appendix A: MAX+PLUS II Command-Line Mode describes how to operate the MAX+PLUS II Compiler, Timing Analyzer, and Simulator in batch mode from the command prompt under UNIX, Microsoft Windows NT, and Microsoft Windows 95.

Appendix B: Altera Support Services explains how to contact Altera's user support services.

Appendix C: Additional UNIX Workstation Configuration Information describes how to change additional UNIX workstation configuration items that control the appearance of MAX+PLUS II windows, serial port configuration, screen height and width, and printer and screen fonts.

Glossary

Index

Section

1

MAX+PLUS II Installation

This section describes how to install MAX+PLUS II software and programming hardware on PCs and UNIX workstations.

Installing MAX+PLUS II on PCs

If you are installing MAX+PLUS II on one or more PCs, read the instructions in the following sections:

Installing MAX+PLUS II on UNIX Workstations

If you are installing MAX+PLUS II on one or more UNIX workstations, read the instructions in the following sections:

If you have any unanswered questions about MAX+PLUS II installation after reading this section or the **read.me** file, contact the Altera Applications Department:

Altera Corporation
Applications Department
101 Innovation Drive
San Jose, CA 95134
Telephone: (800) 800-EPLD (6:00 a.m. to 6:00 p.m. Pacific Time)
or (408) 544-7000 (7:30 a.m. to 5:30 p.m. Pacific Time)
Fax: (408) 544-6401

Go to *Appendix B: Altera Support Services* on page 281 for more information about contacting Altera for technical support, literature, and non-technical customer service.

The read.me File

The MAX+PLUS II **read.me** file provides up-to-date information on installation and operating requirements, including disk space and memory. You should read the **read.me** file before installing the software or hardware.

The **read.me** file is available in the top-level directory of the MAX+PLUS II CD-ROM. After installation, you can open the **read.me** file from the MAX+PLUS II Help menu.

Registering MAX+PLUS II Software

Registering your MAX+PLUS II software is necessary in order to receive future update information. In addition, for some MAX+PLUS II development systems, you must register your software in order to receive an authorization code that allows you to use the software.

You can register your MAX+PLUS II software in one of four ways, depending on the features included in your system and the hardware platform:

- Visit the Altera world-wide web site at **http://www.altera.com**.
- Register the first time you run the MAX+PLUS II software: a Fax Registration Form appears automatically to allow you to register. You can also access this form at any later time by choosing the **Register** button in the **Authorization Code** dialog box (Options menu) in MAX+PLUS II.
- Fill out the registration card attached to the "STOP, PLEASE READ THIS FIRST" card that is included with your MAX+PLUS II system.
- Fill out the "Workstation Registration and License File Request Form" included with your software to both register your software and obtain the license file required to install and run MAX+PLUS II for UNIX workstations.

  This form asks you for the ID of the license server. The license server is the computer that will run the two Altera license daemons (**lmgrd** and **alterad**).

  To determine the ID of the license server, type one of the following commands:

License Server Platform:	Command:
Sun SPARCstation running SunOS 4.1.3+	`# /usr/bin/hostid ↵`
Sun SPARCstation running Solaris 2.5+	`# /usr/ucb/hostid ↵`
HP 9000 Series 700/800 workstation running HP-UX 10.20+ (transpose the resulting value into hexadecimal to determine your license server ID)	`# /usr/bin/uname -i ↵`

License Server Platform:	Command:
IBM RISC System/6000 workstation running AIX 4.1+ (ignore the last two digits of the ID that is displayed and use the remaining 8 digits as the license server ID)	`# /usr/bin/uname -m↵`

The FLEXlm licensing scheme allows either one or three license servers on a network. A single PC or UNIX workstation can function as the file server, license server, and user computer. The file server, license server, and user computer can also be separate UNIX workstations or PCs.

Go to "Configuring Network Licensing" on page 33 for more information about configuring the license server.

Go to *Appendix B: Altera Support Services* on page 281 for more information about contacting Altera technical support services.

Installing MAX+PLUS II on a PC

The following instructions describe the requirements and procedures for installing the MAX+PLUS II software on a PC or compatible computer running Microsoft Windows 95 or Windows NT. This section covers the following topics:

- System requirements for PCs
- Installing MAX+PLUS II software
- Additional Windows NT Installation Steps
- Additional NEC 9801 Installation Steps

Go to the ***User's Guide*** for your version of Microsoft Windows for information on optimizing your system to run MAX+PLUS II.

Go to the MAX+PLUS II **read.me** file for information on installing MAX+PLUS II software on PCs running other versions of the Windows operating system.

System Requirements for PCs

Your system must meet the following minimum requirements:

- Pentium class or higher- (recommended) or 486-based PC
- Microsoft Windows NT version 3.51 or 4.0 (recommended) or Microsoft Windows 95
- Microsoft Windows-compatible graphics card and monitor
- CD-ROM drive
- Microsoft Windows-compatible 2- or 3-button mouse
- Full-length 8-bit slot for the programming card
- Parallel port

Go to the **read.me** file for specific information about disk space and memory requirements in the current version of MAX+PLUS II.

Go to the ***User's Guide*** for your version of Microsoft Windows for more information about fonts.

Installing MAX+PLUS II Software

The following instructions guide you through the installation of MAX+PLUS II on the PC. The following topics are covered:

Determining Free Disk Space

The **Install** program calculates whether you have enough free disk space for installation. The actual space required depends on the allocation unit size on your hard disk, which you can determine by typing `chkdsk` ↵ at a DOS prompt. Refer to the **read.me** file for precise information on the amount of hard disk space required for installation.

Regardless of which drive you select for installing the MAX+PLUS II software, the **Install** program requires at least 500 Kbytes of free hard disk space on your **c:** drive.

Installing the Software

The following steps describe first-time installation with the Altera **Install** program. Once you have completed the following installation procedure, you can run MAX+PLUS II programs directly from your hard disk.

Do not install MAX+PLUS II by copying files directly from the CD-ROM. The installation files are in compressed format and must be decompressed by the **Install** program.

These installation instructions assume the following conditions:

- Your hard disk drive is **c:**.
- You have already installed Windows NT 3.51 or 4.0 or Windows 95 in the **windows** directory.

During installation, an internal error can occur if the portion of the PATH variable in your **autoexec.bat** file that specifies the location of your Windows directory does not include a disk drive letter. For example, a path of **\windows** will cause an internal error. You must edit the PATH variable to avoid this error.

To install the software, perform the following steps:

1. If you are using anti-virus software packages such as the **vsafe.com** anti-virus software provided with MS-DOS version 6.2 or the **Disk Protect** anti-virus software provided with Norton Utilities, Altera recommends that you disable these and other similar software packages in your **autoexec.bat** file before installing MAX+PLUS II.

2. Boot the computer from the hard disk and start Windows.

3. Insert the MAX+PLUS II CD-ROM into a caddy if necessary, insert it into your CD-ROM drive, and perform one of the following:

   ✓ (For Windows NT 3.51) Choose **Run** from the Windows Program Manager File menu and type *<CD-ROM drive>*:`\pc\maxplus2\install` ↵ in the *Command Line* box.

   or:

   ✓ (For Windows NT 4.0 or Windows 95) Choose **Run** from the Windows Start menu and type *<CD-ROM drive>*:`\pc\maxplus2\install` ↵ in the *Open* box.

   The MAX+PLUS II **Install** program starts. This program prompts you for all information needed to install MAX+PLUS II. You should install MAX+PLUS II in a separate directory from any previous MAX+PLUS II version. If your DOS path includes an existing MAX+PLUS II directory, the **Install** program will default to installing in that directory, so be sure to override the default directory during installation.

Because improvements and other changes in the latest version of MAX+PLUS II software may fit projects differently from other versions, you may wish to finish existing projects with the earlier version. Altera also recommends that you archive a project before upgrading the latest version of MAX+PLUS II so that you can return it to an earlier version, if necessary.

4. Follow the directions provided on-screen. To get help on any step in the installation process, choose the **Help** button.

   ☞ You can abort the installation procedure at any time by choosing the **Exit** button.

5. If you wish, you can choose to leave the MAX+PLUS II Help files on the CD-ROM and have MAX+PLUS II access them from there. To have MAX+PLUS II access the Help files from the CD-ROM, follow these steps during installation:

   a. Choose **Custom Installation** in the **MAX+PLUS II Installation Options** dialog box.

   b. Turn off *MAX+PLUS II Help* under *Optional MAX+PLUS II Features*, and choose **Install**.

   c. After the **Install** program has finished, insert the following line in the `[system]` section of the **maxplus2.ini** file in your MAX+PLUS II system directory:

      `HELP_FILE_DIR=`*<CD-ROM drive>*`:\help` ↵

6. During installation of MAX+PLUS II on a PC running Windows 95, you may receive an error message that states that the file **a.azp** was not extracted. To correct this error, disable write caching for SmartDrive by typing the following command at a DOS prompt, then reinstall MAX+PLUS II:

   ```
   smartdrv /x ↵
   ```

7. If you receive an application error message while installing MAX+PLUS II on a Novell network, perform the following steps:

   a. Reboot the computer, but do not log onto the network.

   b. Reinstall MAX+PLUS II according to the installation instructions.

   c. Reboot the computer and log onto the network.

8. Once you have successfully installed MAX+PLUS II, the **read.me** file appears automatically. You should check the **read.me** file for important information on the MAX+PLUS II software. After checking the **read.me** file, exit from Windows.

9. If you are using Windows 95, edit the setting for the `files` variable in your **config.sys** file, which is usually located in the top-level directory of your **c:** drive, as follows:

   ```
   files=50 ↵
   ```

10. The **Install** program may modify your **autoexec.bat** file, which also is usually in the top-level directory of your **c:** drive, to make MAX+PLUS II run properly. You should examine this file to check whether it is compatible with other software on your system. The **Install** program saves the original file as **autoexec.bak**.

11. Remove the CD-ROM from the drive and reboot your computer before starting MAX+PLUS II.

☞ See "MAX+PLUS II File Organization" on page 69 for a description of the directories and subdirectories created during installation.

You can uninstall MAX+PLUS II by following these steps, beginning with step 1 and choosing the **Uninstall** button instead of the **Install** button.

Go to "Installing the Adobe Acrobat Reader" on page 51 for information on installing the Adobe Acrobat Reader that is needed to read the Altera-provided Software Interface Guides for Cadence, Mentor Graphics, Synopsys, and Viewlogic (Powerview) EDA tools.

Additional Windows NT Installation Steps

The following sections guide you through additional steps for installing MAX+PLUS II on the PC. The following topics are covered:

- Installing Windows NT Drivers
- Disabling Floating-Point Emulation
- Using MAX+PLUS II with NTFS

Installing Windows NT Drivers

On computers running Windows NT 3.51 or 4.0, you must install Windows NT drivers to support MAX+PLUS II after you run the **Install** program. The Sentinel driver is required for all MAX+PLUS II systems, and is automatically installed with MAX+PLUS II. The Altera Programmer driver

is required for systems that include the LP6 Logic Programmer Card; and the Altera ByteBlaster driver is required for systems that include the ByteBlaster. Both the Altera Programmer driver and the Altera ByteBlaster driver must be installed separately from MAX+PLUS II.

You do not need to install the Logic Programmer Card or the ByteBlaster before installing the Altera Programmer or Altera ByteBlaster drivers.

To install the Altera Programmer driver, follow these steps:

1. (Windows NT 3.51 only) Double-click Button 1 on the **Drivers** icon in the Control Panel, then choose the **Add** button to open the **Add** dialog box.

   or:

   (Windows 4.0 only) Double-click Button 1 on the **Multimedia** icon in the Control Panel, click Button 1 on the **Devices** tab, and choose the **Add** button to open the **Add** dialog box.

2. Select *Unlisted or Updated Driver* from the *List of Drivers* list box and choose **OK**. The **Install Driver** dialog box opens.

3. Type or select `\<`*MAX+PLUS II system directory*`>\drivers` in the text box, and choose **OK**. The **Add Unlisted or Updated Driver** dialog box opens.

4. Select *Altera Programmer* from the list box and choose **OK**. The **Altera Programming Hardware Setup** dialog box opens.

5. Select the appropriate bus type from the *Bus Type* list box. If you do not know the correct bus type, select *(E)ISA*.

6. Select an unused I/O base address in your PC for your LP6 Logic Programmer Card from the *Physical Address* drop-down list box. The Programmer Card uses 16 contiguous I/O addresses, starting at the selected base address. Altera Programmer Cards are shipped with the default address 280 hex. For more information about changing the I/O address of the Programmer Card, see "Changing the LP6 Card Address Location" on page 56.

7. To install the driver at the current address, choose **OK**. The **System Setting Change** dialog box is displayed.

Your Logic Programmer Card will operate only at the specified address.

If necessary, you can change the driver's I/O address.

To change the driver's I/O address in Windows NT 3.51, follow these steps:

a. Double-click button 1 on the **Drivers** icon in the Windows NT Control Panel. The **Drivers** dialog box opens.

b. Select *Altera Programmer* from the *Installed Drivers* list box, then choose the **Setup** button to open the **Altera Programmer Driver Setup** dialog box and edit the driver's I/O address.

To change the driver's I/O address in Windows NT 4.0, follow these steps:

a. Double-click Button 1 on the **Multimedia** icon in the Windows NT 4.0 Control Panel. The **Multimedia Properties** dialog box opens.

b. Click Button 1 on the **Devices** tab, select *Altera Programmer* under *Other Multimedia Devices*, and then choose the **Properties** button. The **Altera Programmer Properties** dialog box opens.

c. Choose **Settings** to open the **Altera Programming Hardware Setup** dialog box and edit the driver's I/O address.

8. If you wish to install another driver (e.g., for the ByteBlaster), choose the **Don't Restart Now** button in the **System Setting Change** dialog box and repeat the steps above to install another driver. Otherwise, choose the **Restart Now** button in the **System Setting Change** dialog box to reboot your computer.

To install the Altera ByteBlaster driver in Windows NT 3.51 or NT 4.0, follow these steps:

1. Repeat steps 1 through 3 above.

2. Select *Altera ByteBlaster* from the list box and choose **OK**.

3. To install the driver at the current address, choose **OK**. The **System Setting Change** dialog box is displayed.

4. Repeat step 8 above.

Disabling Floating-Point Emulation

If you are running MAX+PLUS II under Windows NT 3.51 or 4.0, Altera recommends that you turn floating-point emulation off to improve timing Simulator Netlist File (**.snf**) extraction time. Type the following command at the command prompt:

```
pentnt -o↵
```

If you do not have the **pentnt.exe** program, contact Microsoft. (In Windows NT 3.51, the **pentnt.exe** program is normally located in the **system32** subdirectory of your Windows NT directory.)

If you are running MAX+PLUS II under another Windows operating system, there is no user control over floating-point emulation.

Using MAX+PLUS II with NTFS

If MAX+PLUS II is installed on a PC running Windows NT 3.51 or 4.0 with NTFS, the **maxplus2.ini** file should be readable and writable by all users, or each user should have his or her own copy of the file. Each user must use the **System** control available in the Windows NT Control Panel to set the `MAXPLUS2_INI` environment variable equal to the drive and directory containing the appropriate **maxplus2.ini** file. See "Creating & Using a Local Copy of the **maxplus2.ini** File" on page 67.

Additional NEC 9801 Installation Steps

On NEC 9801 computers, you must modify the **\windows\system.ini** file after you run the **Install** program for MAX+PLUS II. Add the following lines to the **\windows\system.ini** file:

```
[sentinel] ↵
MACHINE=NEC9800 ↵
```

Installing MAX+PLUS II on a UNIX Workstation

The following instructions describe the requirements and procedures for installing the MAX+PLUS II software on Sun SPARCstations running SunOS 4.1.3+ or Solaris 2.5.+, HP 9000 Series 700/800 workstations, and IBM RISC System/6000 workstations. This section covers the following topics:

System Requirements for UNIX Workstations

The hardware and software system requirements listed here and in the **read.me** file must be met before you can install MAX+PLUS II on a UNIX workstation.

Hardware Requirements for UNIX Workstations

MAX+PLUS II for UNIX workstations requires the following minimum hardware configuration:

- One of the following workstations:
  - Sun Microsystems SPARCstation or compatible workstation
  - HP 9000 Series 700/800 workstation
  - IBM RISC System/6000 workstation
- ISO 9660-compatible CD-ROM drive
- Color monitor

Software Requirements for UNIX Workstations

MAX+PLUS II for UNIX workstations requires the following minimum software configuration:

Table 1-1. UNIX Workstation Software Requirements

System Type	Software Requirements
Sun SPARCstation	SunOS 4.1.3 or higher OpenWindows 3.0 or higher
	Solaris 2.5 or higher
HP 9000 Series 700/800	HP-UX 10.20 or higher
IBM RISC System/6000	AIX 4.1 or higher

Installing the Software & Third-Party Interfaces

MAX+PLUS II installation must be performed by a System Administrator with superuser or "root" privileges. The following steps describe first-time installation with the Altera **install.cd** program.

The installation procedure consists of two phases:

1. Installing the software on the file server
   a. Installing the software and third-party interface files
   b. Configuring the file server and user environments
2. Setting up the licensing software and server

You can complete phase 2 separately, if necessary. When you install an update to MAX+PLUS II, only the first phase of the installation process is required.

The installation instructions assume the following conditions:

- The UNIX environment is case-sensitive. You must enter directory names, filenames, and filename extensions exactly as shown.
- You are logged in as a superuser and you install MAX+PLUS II from a local CD-ROM drive. If not, your System Administrator must provide access to a remote CD-ROM drive.

- The default CD-ROM directory is **/cdrom**.
- MAX+PLUS II will be installed in the **/usr/maxplus2** directory. The installation procedure creates the **maxplus2** directory if it does not already exist.

If you use a different CD-ROM directory or MAX+PLUS II system directory name, substitute the appropriate name in the installation steps.

Boldface text represents text that appears on screen; `Courier` font indicates text you must type. The % character indicates the UNIX C-shell prompt; the # character indicates the superuser prompt. Commands that do not fit on a single line in this manual are indicated by indentations of subsequent lines.

Mounting the CD-ROM

Before installing the software, you must mount the CD-ROM. The commands to do so vary depending on the workstation.

1. Insert the MAX+PLUS II CD-ROM into a caddy if necessary and insert the caddy into your CD-ROM drive.

2. Locate your workstation in Table 1-2 and type the corresponding commands.

Table 1-2. Commands for Mounting the CD-ROM (Part 1 of 2)

Workstation & Operating System	Commands to type
Sun SPARCstation SunOS 4.1.3+	`# mkdir /cdrom` ↵ `# mount -t hsfs -o ro /dev/sr0 /cdrom` ↵ `# cd /cdrom` ↵
Sun SPARCstation Solaris 2.5+	If you are running Volume Manager, the CD-ROM will be mounted automatically as **/cdrom/cdrom0**, and only the last command below is needed. Otherwise, type all three of the following commands: `# mkdir /cdrom/cdrom0` ↵ `# mount -F ufs -r /dev/dsk/c0t6d0s2 /cdrom` ↵ `# cd /cdrom/cdrom0` ↵

Table 1-2. Commands for Mounting the CD-ROM (Part 2 of 2)

Workstation & Operating System	Commands to type
HP 9000 Series 700/800	`# mkdir /cdrom` ↵ `# /etc/mount -t cdfs /dev/dsk/`*<SCSI ID of the CD-ROM drive>*`s0 /cdrom` ↵ `# cd /cdrom` ↵
IBM RISC System/6000	`# mkdir /cdrom` ↵ `# crfs -v cdrfs -p ro -dcd0 -m /cdrom -Ano -tn` ↵ `# mount -v cdrfs -r /dev/cd0 /cdrom` ↵ `# cd /cdrom` ↵

Running the Installation Program

The installation program has several phases, each of which can be completed separately. These phases are as follows:

- Starting the installation program
- Installing the network licensing file
- Installing the third-party interface files

You can run the installation program as many times as necessary to complete your installation.

Starting the Installation Program

To start the MAX+PLUS II installation program, type `./install.cd` ↵ at the # prompt.

or:

If you mounted the CD-ROM on an HP 9000 Series 700/800 workstation, type `./INSTALL.CD\;1` ↵ at the # prompt.

You can quit the installation procedure at any time by typing Ctrl+C ↵.

The following text is displayed:

MAX+PLUS II Workstation Installation
Copyright (c) Altera Corporation 1997

Type Ctrl+C <Return> to quit installation at any time.

Type the full pathname of the directory where the MAX+PLUS II CD-ROM is mounted (default: /cdrom):

Press ↵ if the MAX+PLUS II CD-ROM is mounted in the default directory. Otherwise, type the correct directory name, then press ↵.

Would you like to install the MAX+PLUS II system files (y/n):

Type `n` ↵ to skip installation of the system files. Type `y` ↵ to install the MAX+PLUS II system files. The following prompt is displayed:

Type the full pathname of the system directory where MAX+PLUS II will be installed (default: /usr/maxplus2):

Press ↵ to accept the default directory. Otherwise, type the name of the desired directory and press ↵.

You may install MAX+PLUS II on one or more of the following platforms:

Platform	System Type	Operating System
sunos	**Sun SPARCstation**	**SunOS 4.1.3+ (Solaris 1.x)**
solaris	**Sun SPARCstation**	**Solaris 2.5+ (SunOS 5.5+)**
hp	**HP 9000 Series 700/800**	**HP-UX 10.20+**
rs6000	**IBM RISC System/6000**	**AIX 4.1+**

Enter one or more of the platform names listed above
(choices are: sunos, solaris, hp, rs6000)
(default: solaris):

Press ↵ to accept the default platform. Otherwise, type the name(s) of the desired platform(s), then press ↵. To install MAX+PLUS II for multiple platforms simultaneously, type multiple platform names separated by spaces, e.g., `solaris hp` ↵.

If you type multiple platform names at the prompt, the following prompt appears:

Enter license server platform type
(choose one of: solaris sunos)
(default: solaris):

Press ↵ to accept the default platform. Otherwise, type the name of the desired license server platform, then press ↵.

The MAX+PLUS II Help files are available on the installation CD-ROM in the /cdrom/help directory, although Altera recommends installing the Help files in your MAX+PLUS II system directory.

Would you like to install the MAX+PLUS II Help files? (y/n):

Type y ↵ to install the Help files. If you want to leave the Help files on the CD-ROM and run them from there, type n ↵.

If you type n ↵, the installation program inserts the following line in the `[system]` section of your **/usr/maxplus2/maxplus2.ini** file to allow MAX+PLUS II to access the Help files from the CD-ROM:

`HELP_FILE_DIR=`*<CD-ROM path>*`:/help ↵`

Would you like to install or modify your network license file? (y/n)

If you type y ↵, you will be prompted for license file information during the installation phase. If you type n ↵, the installation program skips the license file installation. You can also install a network license file manually, as described in "Installing the Network Licensing File" on page 21.

Would you like to install the MAX+PLUS II Sample/Tutorial files? (y/n)

Type n ↵ to skip installation of the MAX+PLUS II Sample/Tutorial files.

Type y ↵ to install the sample files and the files for the **chiptrip** tutorial described in *Section 3: MAX+PLUS II Tutorial.*

If you type y ↵, the following prompt is displayed:

Type the full pathname of the working directory where the MAX+PLUS II Sample/Tutorial files will be installed (default: /usr/max2work):

Press ↵ to accept the default directory. Otherwise, type the name of the desired directory and press ↵.

Would you like to install third-party interfaces? (y/n)

If you selected multiple platforms at the earlier prompt and type y ↵, the following prompt is displayed:

Enter third-party installation platform type
(choose one of: solaris sunos)
(default: solaris):

Press ↵ to accept the default platform. Otherwise, type the name of the desired third-party installation platform, then press ↵.

Some or all of the following information is displayed:

CD-ROM directory:	*<CD-ROM directory name>*
Install system files:	*<yes or no>*
MAX+PLUS II system directory:	*<system directory name>*
Platforms to install:	*<selected platform(s)>*
License server platform:	*<selected platform>*
Install Help:	*<yes or no>*
Install/modify license file:	*<yes or no>*
Install sample/tutorial files:	*<yes or no>*
MAX+PLUS II working directory:	*<working directory name>*
Install third-party interfaces:	*<yes or no>*
Third-party interface platform:	*<selected platform>*

Is this information correct? (y/n):

Type y ↵ if the information is correct. Type n ↵ to change any incorrect item(s). You will be given the option to restart the installation program.

If you type y ↵, the installation program deletes any existing files from the specified MAX+PLUS II system and working directories, checks to see that disk requirements are met, and then installs the items you selected.

If you chose to install the network licensing file, go to "Installing the Network Licensing File," next. If you chose to install the third-party interfaces, go to "Installing the Third-Party Interface Files" on page 24 and "Installing the Adobe Acrobat Reader" on page 51 for information on installing the Adobe Acrobat Reader that is needed to read the Altera-provided Software Interface Guides for Cadence, Mentor Graphics, Synopsys, and Viewlogic (Powerview) EDA tools. Otherwise, go to "Unmounting the CD-ROM" on page 25.

Go to "MAX+PLUS II File Organization" on page 69 for information about the directory structure of the files installed for MAX+PLUS II.

Installing the Network Licensing File

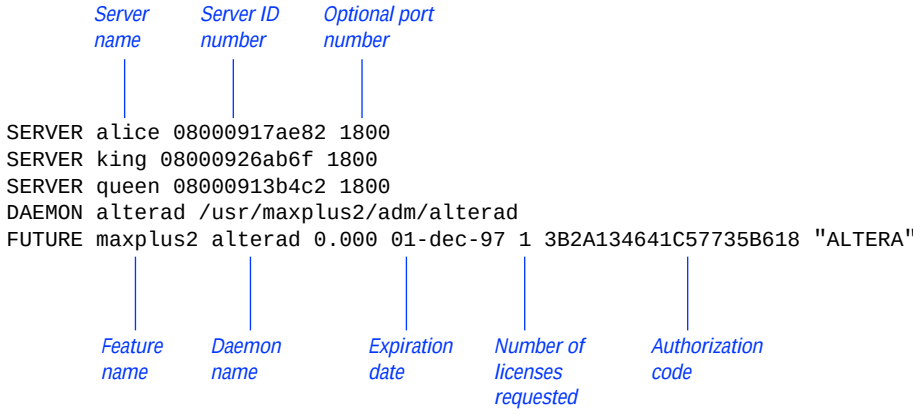
You can install the **license.dat** file needed for network licensing in one of two ways:

- Manually create a copy of the **license.dat** file
- Run the **install.cd** program

When you register your MAX+PLUS II system, Altera provides you with a "Node Authorization" form that includes the license file required to run MAX+PLUS II on your workstation. Go to "Registering MAX+PLUS II Software" on page 4 for instructions.

Figure 1-1 shows a sample license file.

Figure 1-1. Sample License File



If you receive a **license.dat** file from Altera, go through the following steps:

✓ Type or save the file as **/***<MAX+PLUS II system directory>***/adm/ license.dat**. If you already have a **license.dat** file, edit the existing file to add the new Altera-provided information.

or:

✓ When the installation program displays the following prompt, type y ↵.

Would you like to install or modify your network license file? (y/n)

The installation program displays the following prompt:

MAX+PLUS II License File Installation
Copyright (c) Altera Corporation 1997

Type the full pathname of the system directory where MAX+PLUS II has been installed (default:/usr/maxplus2)

Press ↵ to accept the default directory. Otherwise, type the name of the desired directory and press ↵.

Choose one of the following options:

1. **Create a new license**
2. **Update an existing license**

If you type 1 ↵, the following prompt is displayed (if you type 2 ↵, a shortened version of the same prompt is displayed):

The following information is required to install the network licensing file:

1. **The number of license servers at your site**
2. **The host name and host ID for each license server**
3. **The MAX+PLUS II product being licensed**
4. **The maximum number of users that MAX+PLUS II will support concurrently**
5. **The license expiration date**
6. **The authorization code from Altera Customer Service**

Do you want to continue? (y/n)

If you type y ↵, you are prompted to enter the licensing information provided by Altera. To quit license installation, type n ↵.

You can read the information from the Altera-provided **license.dat** file to obtain the information needed to respond to the installation program prompts on the features, expiration date, number of licenses, and authorization code.

Enter the number of license servers: (1/3) *<number of license servers>* ↵

You are asked for the host name and ID for each license server:

Enter the host name for the license server: *<host name for license server>* ↵

Enter the host ID for the license server: *<host ID number>* ↵

Which feature do you wish to install? Choose one:

1. maxplus2 (MAX+PLUS II Base System)
2. maxplus2vhdl (VHDL Synthesis)
Q. Quit

(Choose 1, 2, or Q):

<feature number> ↵

Enter the software expiration date [*<default expiration date>***]:**

<expiration date> ↵

Enter the maximum number of users: *<number of users>* ↵

Enter the authorization code: *<authorization code>* ↵

Is the information correct? (y/n/q)

If you think the information you entered may not be correct, type n ↵. The license file installation starts again from the beginning, to allow you to accept or change each of your original responses.

To quit the license installation, type q ↵.

To accept the information as correct, type y ↵. The following messages are displayed:

Your license file has been created. It is located in /<*MAX+PLUS II system directory*>/adm/license.dat.

The MAX+PLUS II license installation is complete.

If you chose to install the third-party interfaces, go to "Installing the Third-Party Interface Files," next. Otherwise, go to "Unmounting the CD-ROM" on page 25.

Installing the Third-Party Interface Files

If you chose to install the third-party interface files, the installation program displays the following prompt:

Third-Party Interfaces Installation:

1.	**Cadence-Composer**	**x**	**Mbytes**
2.	**Cadence-Concept**	**x**	**Mbytes**
3.	**Mentor Graphics**	**x**	**Mbytes**
4.	**Synopsys**	**x**	**Mbytes**
5.	**Viewlogic**	**x**	**Mbytes**
6.	**All**		
7.	**Quit**		

Enter one or more numbers:

For example, to install both Synopsys and Viewlogic files, you must type 4, 5 ↵ or 4 5 ↵. To quit installation, type 7 ↵.

Once the installation of the third-party interface files is complete, the following message is displayed:

Third-party interface installation is complete.

Go to "Installing the Adobe Acrobat Reader" on page 51 for information on installing the Adobe Acrobat Reader that is needed to read the Altera-provided Software Interface Guides for Cadence, Mentor Graphics, Synopsys, and Viewlogic (Powerview) EDA tools. Otherwise, if you have finished installing MAX+PLUS II software, go to "Unmounting the CD-ROM," next.

Go to the Altera-provided Software Interface Guide for your third-party environment for information on the directory structure of the files installed for third-party interfaces to MAX+PLUS II.

Unmounting the CD-ROM

To unmount the CD-ROM, type `umount /cdrom` ↵ at the # prompt.

Go to "Configuring the File Server & User Environment," next, to continue the installation process.

Configuring the File Server & User Environment

You now must set up the file server and user environment to run MAX+PLUS II. The installation steps vary depending upon the computer platform.

For each platform you are configuring, follow these steps:

1. Configure the file server environment so that the MAX+PLUS II software installed on the file server is available to other computers on the network.

2. Configure the user (client) environment so that users can find the MAX+PLUS II software installed on the file server, and the MAX+PLUS II software can find the license server.

Continue to one of the following sections for instructions:

Configuring a SPARCstation Running SunOS 4.1.3+

This section describes the steps necessary to configure a Sun SPARCstation running SunOS 4.1.3 or higher.

Configuring the File Server

If a single workstation functions as the file server, license server, and user workstation, skip this section and go to "Configuring the User Workstation," next.

If the file server, license server, and user workstations are separate workstations, use the Network File System (NFS) to export the directory that contains MAX+PLUS II. To export the directory, follow these steps:

1. If a higher-level directory in the partition is not already exported, add the line `/usr/maxplus2 ↵` to the **/etc/exports** file on the file server.

2. Save the changes.

3. Export the file by typing `/etc/exportfs -a ↵` at the # prompt.

Configuring the User Workstation

To configure the user workstation, follow these steps:

1. If the file server and user workstations are separate workstations, mount the **/usr/maxplus2** directory on the user workstation with NFS by typing the following commands:

   ```
   # mkdir /usr/maxplus2 ↵
   # /etc/mount <file server name>:/usr/maxplus2 /usr/
     maxplus2 ↵
   ```

2. Update the `PATH` and `LM_LICENSE_FILE` variables that are specified in the **.cshrc** file located in each user's home directory. You must edit this file for each user, or provide clear instructions that describe which lines need to be entered or edited.

   a. Update each user's `PATH` environment variables to include **/usr/maxplus2/bin**:

   ```
   set path = (/usr/local/bin /usr/maxplus2/bin) ↵
   ```

b. Each user must have an LM_LICENSE_FILE variable that is set to the full directory pathname of the license file. To update this variable, add the following line to the **.cshrc** file for each user:

```
setenv LM_LICENSE_FILE /usr/maxplus2/adm/
   license.dat ↵
```

If more than one application uses this environment variable, separate the different paths with a colon (:). For example:

```
setenv LM_LICENSE_FILE /usr/maxplus2/adm/
   license.dat:/tmp/license.xyz ↵
```

3. After saving the changes to each user's **.cshrc** file, type the following commands:

```
# cd ↵
# source .cshrc ↵
```

Go to "Configuring Network Licensing" on page 33 for information about configuring the license server.

Configuring a SPARCstation Running Solaris 2.5+

This section describes the steps necessary to configure a Sun SPARCstation running Solaris 2.5 or higher.

Configuring the File Server

If a single workstation functions as the file server, license server, and user workstation, skip this section and go to "Configuring the User Workstation," next.

If the file server, license server, and user workstations are separate workstations, use the Network File System (NFS) to export the directory that contains MAX+PLUS II.

To export the directory, follow these steps:

1. If a higher-level directory in the partition is not already exported, add the following line to the **/etc/dfs/sharetab** file on the file server:

```
/usr/maxplus2 "MAX+PLUS II" nfs rw "MAX+PLUS II
   System Directory" ↵
```

2. Save the changes.

3. Export the file by typing `shareall` ↵ at the # prompt.

Configuring the User Workstation

To configure the user workstation, follow these steps:

1. If the file server and user workstations are separate workstations, mount the **/usr/maxplus2** directory on the user workstation with NFS by typing the following commands:

```
# mkdir /usr/maxplus2 ↵
# /etc/mount <file server name>:/usr/maxplus2 /usr/
   maxplus2 ↵
```

2. Update the `PATH` and `LM_LICENSE_FILE` variables that are specified in the **.cshrc** file located in each user's home directory. You must edit this file for each user, or provide clear instructions that describe which lines need to be entered or edited.

   a. Update each user's `PATH` environment variables to include **/usr/maxplus2/bin**:

```
set path = (/usr/local/bin /usr/maxplus2/bin) ↵
```

   b. Each user must have an `LM_LICENSE_FILE` variable that is set to the full directory pathname of the license file. To update this variable, add the following line to the **.cshrc** file for each user:

```
setenv LM_LICENSE_FILE /usr/maxplus2/adm/
   license.dat ↵
```

   If more than one application uses this environment variable, separate the different paths with a colon (`:`). For example:

```
setenv LM_LICENSE_FILE /usr/maxplus2/adm/
   license.dat:/tmp/license.xyz ↵
```

3. After saving the changes to each user's **.cshrc** file, type the following commands:

```
# cd ↵
# source .cshrc ↵
```

Go to "Configuring Network Licensing" on page 33 for information about configuring the license server.

Configuring an HP 9000 Series 700/800 Workstation

This section describes the steps necessary to configure an HP 9000 Series 700/800 workstation.

Configuring the File Server

If a single workstation functions as the file server and user workstation, skip this section and go to "Configuring the User Workstation."

If the file server and user workstations are separate workstations, use the Network File System (NFS) to export the directory that contains MAX+PLUS II. To export the directory, follow these steps:

1. Type `sam` ↵ at the # prompt to bring up the System Administration Manager (SAM).
2. Choose **NFS (Network File Systems) Configuration** from the File Systems Management menu.
3. Set options to allow remote (NFS) file systems access to the local file system's **/usr/maxplus2** directory.
4. In order to process large projects, Altera recommends configuring the file server to allow processes to use more than 64 Mbytes of memory.

   ✓ If you are using SAM, follow these steps:

   a. Double click Button 1 on the **Kernel Configuration** icon. The **Kernel Configuration** window opens.

   b. Double-click Button 1 on the **Configurable Parameters** icon. The **Configurable Parameters** list box appears.

   c. Double-click Button 1 on `maxdsize` to select it from the list. The **Modify Configuration Parameter** *<machine name>* dialog box opens.

   d. Specify a value in the *Formula/Value* box and choose **OK**.

e. In the **Kernel Configuration** window, choose **Create a New Kernel** (Actions menu) and choose **Yes** or **No** in the **Confirmation** dialog box.

If you choose **Yes**, SAM creates the new kernel based on the specified project size and then prompts you to reboot your workstation.

or:

✓ Refer to the documentation on **maxdsize** under *System Parameters* in the ***HP System Administration Tasks Manual*** for information on configuring the file server.

Configuring the User Workstation

To configure the user workstation, follow these steps:

1. If the file server and user workstations are separate workstations, mount the **/usr/maxplus2** directory on the user workstation with NFS by typing the following commands:

```
# mkdir /usr/maxplus2 ↵
# /etc/mount <file server name>:/usr/maxplus2 /usr/
   maxplus2 ↵
```

2. Update the PATH and LM_LICENSE_FILE variables that are specified in the **.cshrc** file located in each user's home directory. You must edit this file for each user, or provide clear instructions that describe which lines need to be entered or edited.

   a. Update each user's PATH environment variables to include **/usr/maxplus2/bin**:

```
set path = (/usr/local/bin /usr/maxplus2/
   bin...) ↵
```

   b. Each user must have an LM_LICENSE_FILE variable that is set to the full directory pathname of the license file. To update this variable, add the following line to the **.cshrc** file for each user:

```
setenv LM_LICENSE_FILE /usr/maxplus2/adm/
   license.dat ↵
```

If more than one application uses this environment variable, separate the different paths with a colon (:). For example:

```
setenv LM_LICENSE_FILE /usr/maxplus2/adm/
   license.dat:/tmp/license.xyz ↵
```

3. After saving the changes to each user's **.cshrc** file, type the following commands:

```
# cd ↵
# source .cshrc ↵
```

Go to "Configuring Network Licensing" on page 33 for information about configuring the license server.

Configuring an IBM RISC System/6000 Workstation

This section describes the steps necessary to configure an IBM RISC System/6000 workstation.

Configuring the File Server

If a single workstation functions as the file server and user workstation, skip this section and go to "Configuring the User Workstation," next.

If the file server and user workstations are separate workstations, use the Network File System (NFS) to export the directory that contains MAX+PLUS II. To export the directory, follow these steps:

1. If a higher-level directory in the partition is not already exported, add the line `/usr/maxplus2 ↵` to the **/etc/exports** file on the file server.

2. Save the changes. The directory will be exported automatically.

Configuring the User Workstation

To configure the user workstation, follow these steps:

1. If the file server, license server, and user workstations are separate workstations, mount the **/usr/maxplus2** directory on the user workstation with NFS by typing the following commands:

```
# mkdir /usr/maxplus2 ↵
# /etc/mount <file server name>:/usr/maxplus2 /usr/
   maxplus2 ↵
```

2. Update the PATH and LM_LICENSE_FILE variables that are specified in the **.cshrc** file located in each user's home directory. You must edit this file for each user, or provide clear instructions that describe which lines need to be entered or edited.

   a. Update each user's PATH environment variables to include **/usr/maxplus2/bin**:

   ```
   set path = (/usr/local/bin /usr/maxplus2/
      bin...) ↵
   ```

   b. Each MAX+PLUS II user must have an LM_LICENSE_FILE variable that is set to the full directory pathname of the license file. To update this variable, add the following line to the **.cshrc** file for each user:

   ```
   setenv LM_LICENSE_FILE /usr/maxplus2/adm/
      license.dat ↵
   ```

   If more than one application uses this environment variable, separate the different paths with a colon (:). For example:

   ```
   setenv LM_LICENSE_FILE /usr/maxplus2/adm/
      license.dat:/tmp/license.xyz ↵
   ```

3. After saving the changes to each user's **.cshrc** file, type the following commands:

   ```
   # cd ↵
   # source .cshrc ↵
   ```

Go to "Configuring Network Licensing" on page 33 for information about configuring the license server.

Refer to "Configuring an IBM RISC System/6000 Workstation Serial Port for Programming" on page 64 for information on the special steps needed to configure an IBM RISC System/6000 workstation to use the BitBlaster.

Configuring Network Licensing

This section provides instructions for configuring network licensing for MAX+PLUS II. This section discusses the following topics:

Configuring the License Server

The FLEXlm licensing scheme allows either one or three license servers on a network. A single UNIX workstation can function as the file server, license server, and user workstation. The file server, license server, and user workstations (or user PCs) can also be separate computers.

To configure the license server, follow these steps:

1. If a single workstation functions as the file server and license server, skip this step and go to step 2.

   If the file server and license server are separate computers, mount the **/usr/maxplus2** directory on the license server with NFS by typing the following commands:

   ```
   # mkdir /usr/maxplus ↵
   # /etc/mount <file server name>:/usr/maxplus2 /usr/
     maxplus2 ↵
   ```

2. Start the license manager daemon on all license servers before starting MAX+PLUS II. Type the following command on each license server:

   ```
   # /usr/maxplus2/adm/max2protd /usr/maxplus2 ↵
   ```

   To invoke the license manager daemon automatically during start-up, add the following lines to the **/etc/rc.local** file on each license server:

```
if [-f /usr/maxplus2/adm/max2protd]; then ↵
su <username> -c "umask 022 ; /usr/maxplus2/adm
/max2protd /usr/maxplus2" ↵
echo -n max2protd ↵
fi ↵
```

3. Type the following command to make the file executable:

```
# chmod +x /etc/rc.local ↵
```

4. If you are configuring an IBM RISC System/6000 as the license server, add the following command to the **/etc/inittab** file after the lines that invoke networking:

```
rclocal:2:wait:/etc/rc.local > /dev/console 2>&1 ↵
```

Troubleshooting License Installation

Most installation errors are caused by improperly installed license daemons (**lmgrd** and **alterad**). If you have completed the procedures described for the software and license installation but the daemons have not started, you must verify that the daemons can be located and have the correct permissions. This section lists the most common error messages that can occur during installation, provides information on possible causes, and suggests corrective actions.

For help with error messages not described here and further assistance with troubleshooting license administration, contact the Altera Applications Department. Go to *Appendix B: Altera Support Services* for more information about technical support.

Message: `/usr/maxplus2/max2protd: Command not found`

Cause: The **max2protd** script or the **lmgrd** or **alterad** daemons cannot be located when you try to start them manually or at system start-up. The **/usr/maxplus2** directory has not been properly mounted with NFS.

Action: Verify that the **/usr/maxplus2** directory, which is exported by the file server with NFS, has been successfully mounted by the license server. This directory should have been mounted when you set up the file server environment, as described in the "Configuring the File Server & User Environment" section for your computer. To

verify that the directory has been mounted correctly, type the following commands on the license server:

```
# cd /usr/maxplus2/adm ↵
# ls -l ↵
```

The **ls** command output must include the following lines, which list the **max2protd**, **lmgrd**, and **alterad** daemons:

```
-rwxr-xr-x 1 root 278528 jun 01 13:03 alterad
-rwxr-xr-x 1 root  81920 jun 01 13:04 lmgrd
-rwxr-xr-x 1 root    568 jun 01 11:02 max2protd
```

The dates and file sizes may be different, but the three named files must be present. If the named files are not displayed, the directory has not been mounted correctly. Refer to the "Configuring the File Server & User Environment" section for your computer for instructions on how to mount the **/usr/maxplus2** directory.

Message: `license daemon: execl failed`

Cause: The **alterad** daemon does not exist or cannot be executed.

Action: Verify the existence of the **alterad** daemon by typing the following commands:

```
# cd /usr/maxplus2/adm ↵
# ls -l ↵
```

The output of the `ls` command must include the following lines, which list the **max2protd**, **lmgrd**, and **alterad** daemons:

```
-rwxr-xr-x 1 root 278528 Jun 01 13:03 alterad
-rwxr-xr-x 1 root  81920 Jun 01 13:04 lmgrd
-rwxr-xr-x 1 root    568 Jun 01 11:02 max2protd
```

If the MAX+PLUS II software cannot find the **alterad** daemon in this directory, reinstall the MAX+PLUS II software as described in "Installing the Software & Third-Party Interfaces" on page 15.

If the **alterad** daemon exists, verify that it can be executed by typing `chmod 755 alterad ↵`.

Message: `Retrying socket bind (address in use)`

Cause: The license daemons are not running because another **lmgrd** license manager daemon is running and using the same TCP/IP port address.

Action: Terminate the daemon processes:

1. Ask all users who use **lmgrd**-based applications to log off.

2. If MAX+PLUS II is installed on a Sun SPARCstation (SunOS 4.1.3 or higher) or an IBM RISC System/6000 workstation, type the following command to determine the license daemon process IDs:

   ```
   # ps waux | grep lmgrd | grep -v grep ↵
   ```

   The response includes the entries for active license manager daemons. For example:

   ```
   root   14803 0 0 0 2 60 76 ? S Jun 01 0:03
      /usr/maxplus2/adm/lmgrd -c /usr/maxplus2
      /adm/license.dat
   ```

   or:

   If MAX+PLUS II is installed on an HP 9000 Series 700/800 workstation or a Sun SPARCstation running Solaris 2.5 or higher, type the following command to determine the license daemon process IDs:

   ```
   # ps -e | grep lmgrd | grep -v grep ↵
   ```

   The response includes the entries for active license manager daemons. For example:

   ```
   11478 ttyp2 0:00 lmgrd
   ```

3. Terminate the license daemons with the following command:

   `kill` *<pid>* ↵

The <*pid*> variable is the process ID number that corresponds to the license daemon. In the example shown in step 2, <*pid*> would be `14803`.

☞ Do not use the `-9` option with the `kill` command.

4. Edit the **/usr/maxplus2/adm/license.dat** file and replace `1800` with a unique four-digit port address that is not used elsewhere on your system and does not conflict with other applications that use **lmgrd**. To replace the port address, change the third field in the `SERVER` line in the **license.dat** file.

   In the following example, you would replace `1800` with a unique four-digit number:

   ```
   SERVER artoo 54321234 1800 ↵
   ```

5. Save the changes to **/usr/maxplus2/adm/license.dat**, then restart the Altera daemon by typing `/usr/maxplus2/adm/max2protd ↵` at the # prompt.

   You should also restart the daemons for other applications.

Message:
```
Starting Altera License daemons
Cannot locate the license manager daemon (lmgrd)
```

Cause: The **max2protd** script or the **lmgrd** or **alterad** daemons cannot be executed from the UNIX prompt or at start-up with the **/etc/rc.local** start-up command on a Sun SPARCstation or IBM RISC System/6000 workstation or with the **/etc/rc** command on an HP 9000 Series 700/800 workstation.

Action: Verify that the **/usr/maxplus2** directory, which is exported by the file server with NFS, has been successfully mounted by the license server. This directory should have been mounted when you set up the file server and the license server environments, as described in the "Configuring the File Server & User Environment" section for your computer. To verify that the directory has been mounted correctly, type the following commands on the license server:

```
# cd /usr/maxplus2/adm ↵
# ls -l ↵
```

The **ls** command output must include the following lines, which list the **max2protd**, **lmgrd**, and **alterad** daemons:

```
-rwxr-xr-x 1 root 278528 Jun 01 13:03 alterad
-rwxr-xr-x 1 root  81920 Jun 01 13:04 lmgrd
-rwxr-xr-x 1 root    568 Jun 01 11:02 max2protd
```

The dates and file sizes may be different, but the three named files must be present. If the named files are not displayed, the directory has not been mounted correctly. Refer to the "Configuring the File Server & User Environment" section for your computer for instructions on how to mount the **/usr/maxplus2** directory.

If the files exist, you must verify that they have the permissions `-rwxr-xr-x`. If the executable bit is not set for **lmgrd** and **alterad**, type the following commands:

```
# cd /usr/maxplus2/adm ↵
# chmod 755 max2protd lmgrd alterad ↵
```

License Administration Options File

The entries in the options file control the operating parameters of the FLEXlm utility. The System Administrator can use these options to reserve licenses, restrict the use of licenses, and define user groups for use with license reservations. The options file can be edited with any text editor. The following options are available:

Option:	Action:
RESERVE	Reserves licenses for a user, host, display, or group of users.
INCLUDE	Includes a user, host, display, or group of users in a list of users who are allowed to use a software feature.
EXCLUDE	Excludes a user, host, display, or group of users from a list of users who are allowed to use a software feature.
GROUP	Defines a user group for use with the RESERVE, INCLUDE, and EXCLUDE options.

Option:	Action:
TIMEOUT	Specifies the time after which an inactive license is returned to the free pool, for use by someone else.
NOLOG	Causes messages of the specified type to be filtered out of the log output for **lmgrd**.

To set any number of options, you must create an options file and list its pathname as the fourth field on the DAEMON line for **alterad**.

An options file consists of lines in the following format:

```
RESERVE <number> <feature> {USER | HOST | DISPLAY | GROUP}
    <name>
INCLUDE <feature> {USER | HOST | DISPLAY | GROUP} <name>
EXCLUDE <feature> {USER | HOST | DISPLAY | GROUP} <name>
GROUP <name> <list of users>
TIMEOUT <feature> <timeout in seconds>
NOLOG {IN | OUT | DENIED | QUEUED}
```

Lines in an options file that begin with the pound character (#) are ignored and can be used as comments.

In the following example, the options file, called **local.options**, reserves a copy of the compile feature for user robert, three copies for user dalia, and a copy for anyone on a computer with the hostname mainline. The file also causes QUEUED messages to be omitted from the logfile and prevents user lori from using the compile feature.

```
RESERVE 1 compile USER robert
RESERVE 3 compile USER dalia
RESERVE 1 compile HOST mainline
EXCLUDE compile USER lori
NOLOG QUEUED
```

If these options are included in the file **/usr/local/flexlm/options/local.options**, the license file DAEMON line must be specified as follows:

```
DAEMON alterad /usr/maxplus2/adm/alterad
   /usr.local.flexlm/options/local.options ↵
```

License Administration FLEXlm Utilities

The following FLEXlm utilities help the System Administrator manage the licensing activities on the network:

- **lmgrd**
- **lmstat**
- **lmdown**
- **lmremove**
- **lmreread**
- **lmver**
- **lmhostid**

lmgrd

The **lmgrd** utility is the main daemon program for FLEXlm. When it is active, it looks for a license file containing all required feature information.

Usage:

`lmgrd [-2] [-b] [-c` *<license file>*`] [-d] [-l` *<log file>*`] [-p]`
`[-s` *<interval>*`] [-t` *<timeout value>*`] [-i` *<feature>*`]`

Option:	Action:
`-2`	Specifies V2 start-up arguments. The `-2` option is the opposite of the `-b` option and is required if you intend to use the `-p` option.
`-b`	Specifies backward compatibility mode. The `-b` option is the default.
`-c` *<license file>*	Uses the specified license file. If this option is not specified, **lmgrd** looks for the environment variable `LM_LICENSE_FILE`. If the environment variable is not set, **lmgrd** looks for **/usr/local/flexlm/licenses/license.dat**.
`-d`	Specifies that hostnames that are read from the license file should have the local domain name appended to them before sending to a client. This option is useful when clients are accessing licenses from another domain.

Option:	Action:
`-l` *<log file>*	Specifies the output log file.
`-p`	Specifies that the **lmdown** and **lmremove** utilities can only be run by a "license administrator." A "license administrator" is a member of the **lmadmin** group, or if the **lmadmin** group does not exist, "license administrator" is a member of group 0.
`-s` *<interval>*	Specifies the log file time-stamp interval, in minutes. The default is 360 minutes.
`-t` *<timeout value>*	Specifies the timeout interval during which daemons must complete their connections to each other. The default is 10 seconds. You can use a larger value if the daemons are running on busy systems or networks.
`-i` *<feature>*	Displays information about the named feature, or, if no feature name is given, displays information about all features.

Altera recommends that you use the `-p` option when starting the **lmgrd** utility to provide security.

See also "**lmdown**" on page 42 and "**lmstat**" on page 41.

lmstat

The **lmstat** utility helps monitor the status of all network licensing activities, including:

- Active daemons
- Users of individual features
- Users of features served by a specific daemon

Usage:

```
lmstat [-a] [-S <daemon>] [-f <feature>] [-s <server>]
   [-t <timeout value>] [-c <license file>] [-A]
   [-l <regular expression>]
```

Option:	Action:
`-a`	Displays information about all features.
`-S` *<daemon>*	Lists all users of the specified daemon's features.
`-f` *<feature>*	Lists all users of the specified feature(s).
`-s` *<server>*	Displays the status of the specified server node(s).
`-t` *<timeout value>*	Specifies the timeout interval during which daemons must complete their connections to each other. The default is 10 seconds. You can use a larger value if the daemons are running on busy systems or networks.
`-c` *<license file>*	Uses the specified license file. If this option is not specified, **lmstat** looks for the environment variable `LM_LICENSE_FILE`. If the environment variable is not set, **lmstat** looks for **/usr/local/flexlm/licenses/license.dat**.
`-A`	Lists all active licenses.
`-l` *<reg. expression>*	Lists all users of the features matching the given regular expression.

See also "**lmgrd**" on page 40.

lmdown

The **lmdown** utility instructs **lmgrd** and **alterad** to shut down. The license daemons write out their last messages to the log file, close the file, and exit. All licenses that have been given out by those daemons are rescinded, so that the next time a program verifies the license, the license is not valid.

Usage:

`lmdown [-c` *<license file>*`]`

The correct license file is in **/usr/maxplus2/adm/license.dat**. The System Administrator should protect the execution of **lmdown**, since shutting down the servers will cause loss of licenses.

Option:	Action:
`-c` *<license file>*	Uses the specified license file. If this option is not specified, **lmdown** looks for the environment variable `LM_LICENSE_FILE`. If the environment variable is not set, **lmdown** looks for **/usr/local/flexlm/licenses/license.dat**. In addition, **lmdown** accepts the `-c` *<license file>* argument that specifies the license file location.

See also "**lmgrd**" on page 40, "**lmstat**" on page 41, and "**lmreread**" on page 44.

lmremove

The **lmremove** utility allows the System Administrator to remove a single user's license for a specified feature. This utility is required if the licensed user was running the software on a node that subsequently crashed, causing the license to become unusable. The **lmremove** utility allows the license to return to the pool of available licenses.

Usage:

`lmremove [-c` *<license file>*`]` *<feature> <user> <host>* `[`*<display>*`]`

Option:	Action:
`-c` *<license file>*	Uses the specified license file. If a license file is not specified, **lmremove** looks for the environment variable `LM_LICENSE_FILE`. If no `-c` option is specified, **lmreread** looks for the environment variable `LM_LICENSE_FILE` to find the license file. If the environment variable is not set, **lmremove** looks for **/usr/local/flexlm/licenses/license.dat**.

See also "**lmstat**" on page 41 and "**lmremove**" on page 43.

lmreread

The **lmreread** utility allows the System Administrator to tell the license daemon to reread the license file and start any new daemons that have been added. This utility is useful if the data in the license file has changed. The new data can be loaded into the license daemon without shutting it down and restarting it. In addition, all pre-existing daemons will be signaled to reread the license file for changes in licensing information.

The **lmreread** utility uses the license filename from the command line (or the default filename, if no license filename is specified) to find the **alterad** daemon. The **lmreread** utility then gives **alterad** the command to reread the license file because the data in the file has changed. The **alterad** daemon always rereads the original file that it loaded. If the path to the license file read by **alterad** must be changed, the System Administrator must shut down **alterad** and restart with the new license file path.

The System Administrator cannot use **lmreread** if the SERVER node names or port numbers have been changed in the license file. In this case, the daemon must be shut down and restarted for the changes to take effect.

The **lmreread** utility does not change any option information specified in an options file. If the new license file specifies a different options file, the information is ignored. If the options file needs to be reread, the System Administrator must shut down the daemon and restart it.

Usage:

`lmreread [-c` *<license file>*`]`

Option:	**Action:**
`-c` *<license file>*	Uses the specified license file. If a license file is not specified, **lmreread** looks for the environment variable, `LM_LICENSE_FILE`. If no `-c` option is specified, **lmreread** looks for the environment variable `LM_LICENSE_FILE` to find the license file. If the environment variable is not set, **lmreread** looks for **/usr/local/flexlm/licenses/license.dat**.

See also “**lmdown**” on page 42.

lmver

The **lmver** utility reports the FLEXlm version of a library or binary file.

Usage:

`lmver` *<filename>*

lmhostid

The **lmhostid** utility reports the host ID of a system.

Usage:

```
lmhostid
```

The following lines show sample output of **lmhostid**:

```
lmhostid-Copyright(C)1989,1990 Highland Software, Inc.
The FLEXlm host ID of this machine is "69021c89"
```

Installing the PC Software Guard

To run MAX+PLUS II software on a PC, you must install the MAX+PLUS II Software Guard and/or the authorization code provided with your development system. If your development system includes a Software Guard, you must install the guard before you can enter the authorization code successfully.

If you are installing MAX+PLUS II software using an ES Site License, you do not need a Software Guard. Go to "Specifying the Authorization Code for a Software Guard Installation" on page 48 for more information.

To install the Software Guard on a PC or compatible computer, follow these steps:

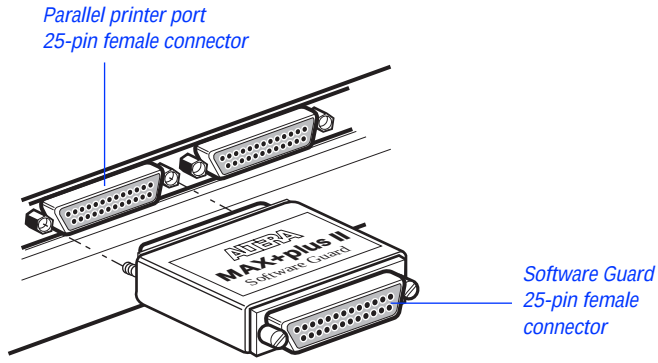
1. Locate a parallel printer port (i.e., an LPT port) on the computer. If you have a printer connected to this port, disconnect it temporarily.

2. Insert the 25-pin male connector end of the Software Guard into the 25-pin female connector of the parallel printer port, as shown in Figure 1-2.

1. Do not connect the MAX+PLUS II Software Guard to either end of an Iomega Zip or Ditto drive. Iomega Zip and Ditto drives can destroy a Software Guard by drawing more power than the guard is capable of handling.

2. Disconnect the Software Guard before using the parallel port with the Interlink file transfer program. Failure to do so can cause damage to the Software Guard.

Figure 1-2. Attaching the Software Guard to a PC



3. If necessary, re-insert the printer cable connector into the female connector of the Software Guard.

4. If you move the MAX+PLUS II Software Guard to a different port, you may need to update the **maxplus2.ini** file to include the name and location of the new port if MAX+PLUS II cannot locate the guard. For example, if you change the Software Guard port to LPT2, ensure that your **maxplus2.ini** file contains the following line:

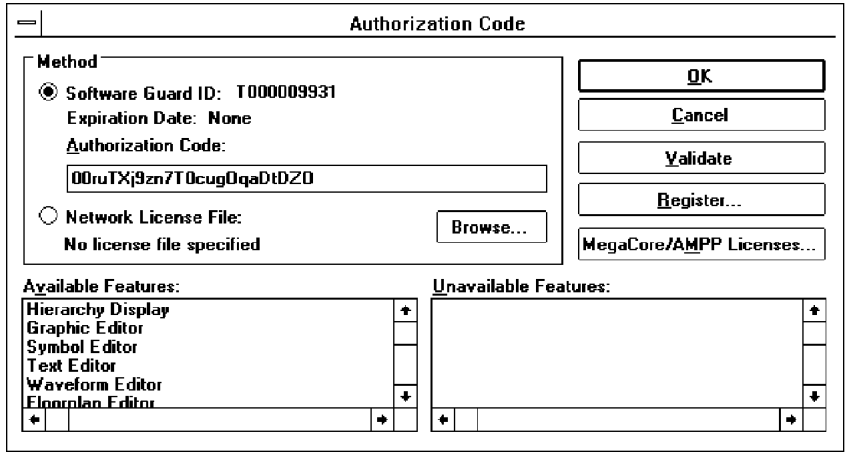
```
GUARD_PORT=2
```

If you suspect that your Software Guard is faulty or damaged, contact the Altera Applications Department. Go to *Appendix B: Altera Support Services* for more information about technical support.

Specifying the Authorization Code or License File

Most development system configurations require you to enter an authorization code when you run MAX+PLUS II for the first time. When you start MAX+PLUS II for the first time, the **Authorization Code** dialog box (Options menu) appears automatically. See Figure 1-3.

Figure 1-3. MAX+PLUS II Authorization Code Dialog Box



The steps required to specify the authorization code depend on whether you have a Software Guard or license file-based MAX+PLUS II system installation.

Specifying the Authorization Code for a Software Guard Installation

To specify the authorization code for an installation that includes a Software Guard, follow these steps:

1. In the **Authorization Code** dialog box, select *Software Guard ID* under *Method*, and verify that the ID number displayed matches the number printed on your Software Guard.

2. Type your authorization code, using upper- and lower-case letters exactly as provided by Altera.

3. Choose **Validate** to confirm your authorization code. Available features will be listed in the *Available Features* box.

4. Choose **OK**.

Specifying the License File for a License File Installation

To specify the license file that contains your authorization code, follow these steps:

✓ In the **Authorization Code** dialog box, select *Network License File* under *Method*. Available features will be listed in the *Available Features* box.

If the dialog box indicates that MAX+PLUS II cannot locate the **license.dat** file, choose **Browse** to open the **License File** dialog box and locate a suitable **license.dat** file.

MAX+PLUS II Site License Information

When you purchase a MAX+PLUS II development system for a single-user PC, it includes a site license to install an unlimited number of copies of the PLS-ES feature set. Feel free to pass MAX+PLUS II software along to your colleagues. The additional MAX+PLUS II systems on the site license have only one requirement: after installation, each individual user must contact Altera for an authorization code (at no cost). For details on getting an authorization code, go to "Registering MAX+PLUS II Software" on page 4.

Specifying Authorization Codes for MegaCore & AMPP Licenses

To specify authorization codes for Altera-provided MegaCore megafunctions or AMPP-provided megafunctions, follow these steps:

1. In the **Authorization Code** dialog box, choose **MegaCore/AMPP Licenses**. The **MegaCore/AMPP Licenses** dialog box is displayed.

2. Type the megafunction ID number provided by Altera or an AMPP vendor in the *Megafunction ID* box.

3. Type the authorization code for your megafunction using upper- and lower-case letters, exactly as provided by Altera or the AMPP vendor, in the *License Authorization Code* box.

4. Choose **Add** to add the megafunction ID number and authorization code to the list of *Existing IDs/Authorization Codes* without closing the dialog box.

   or:

   Choose **OK**.

Once you enter the appropriate information and choose **OK**, you can fully compile the megafunction in MAX+PLUS II to generate output netlist files and programming files.

Installing the Adobe Acrobat Reader

The Altera-provided Software Interface Guides for EDA tools from Cadence, Mentor Graphics, Synopsys, and Viewlogic (Powerview) are provided in the **\lit** directory on the MAX+PLUS II CD-ROM in the Adobe Portable Document File (PDF) format. These files are readable with the Adobe Acrobat Reader 3.0, which requires 4 Mbytes of application RAM. If you do not already have a copy of the Adobe Acrobat Reader, you can install it from the MAX+PLUS II CD-ROM.

To install the Adobe Acrobat Reader on a PC running Windows NT or Windows 95, follow these steps:

1. Insert the MAX+PLUS II CD-ROM into your CD-ROM drive.
2. Choose **Run** (Start menu) and type *<CD-ROM drive>*`:\acroread\win\32bit\setup.exe` ↵.
3. Follow the instructions in the setup program.

To install the Adobe Acrobat Reader on a UNIX workstation, follow these steps:

1. Refer to the **/acroread/instguid.txt** file on the MAX+PLUS II CD-ROM for information on system requirements and compatibility.
2. Ensure that you are logged on as a superuser.
3. Mount the MAX+PLUS II CD-ROM as described in Table 1-2 on page 16.
4. If you have an HP 9000 Series 700/800 workstation, type the following commands:

```
# cd /cdrom/cdrom0/ACROREAD/UNIX ↵
# ./INSTALL\;1 ↵
```

   On other supported UNIX workstation platforms, type the following commands:

```
# cd /cdrom/cdrom0/acroread/unix ↵
# ./install ↵
```

5. Follow the instructions in the install program.

6. To run the Adobe Acrobat Reader, type the following command:

```
# /<installation directory>/bin/acroread ↵
```

Once you have installed the Adobe Acrobat Reader, you can read the following Software Interface Guide files in the **\lit** directory:

Document:	**Filename:**
Cadence & MAX+PLUS II Software Interface Guide	**ac_sig.pdf**
Mentor Graphics & MAX+PLUS II Software Interface Guide	**am_sig.pdf**
Viewlogic Powerview & MAX+PLUS II Software Interface Guide	**av_sig.pdf**
Synopsys & MAX+PLUS II Software Interface Guide	**as_sig.pdf**

Go to the MAX+PLUS II **read.me** file for information on other supported platforms for the Adobe Acrobat Reader.

Installing the Programming Hardware

Table 1-3 shows the programming hardware configurations that are available for use with MAX+PLUS II.

Table 1-3. MAX+PLUS II Programming Hardware Configurations

Platform	Hardware/Application
PC	The LP6 Programmer Card and the Master Programming Unit (MPU) base unit and its adapters.
PC	The FLEX Download Cable, which is used in conjunction with the LP6 Logic Programmer Card, MPU, and a Configuration EPROM programming adapter to download configuration data to FLEX 10K, FLEX 8000, FLEX 6000, and MAX 7000S devices in-system.
PC or UNIX workstation	The BitBlaster Serial Download Cable, which is connected to a serial port to download configuration data to FLEX 10K, FLEX 8000, and FLEX 6000 devices or to program MAX 9000 and MAX 7000S devices in-system.
PC	The ByteBlaster Parallel Download Cable, which is connected to a parallel port to download configuration data to FLEX 10K, FLEX 8000, and FLEX 6000 devices or to program MAX 9000 and MAX 7000S devices in-system.

This section covers the following topics:

Installing PC-Based Programming Hardware

If you have purchased a system with an LP6 Logic Programmer card, you should follow the installation instructions in the order described here.

1. Install the LP6 card by following the instructions that begin on page 54.

2. Assemble the Master Programming Unit (MPU) and connect it to the Logic Programmer Card in your PC by following the instructions below.

3. Optionally connect the FLEX Download Cable to your PC by following the instructions that begin on page 60.

If your MAX+PLUS II system does not include an LP6 card, depending on which programming or configuration technique you plan to use, go through one of the following steps:

- Connect the BitBlaster Serial Download Cable to your PC by following the instructions that begin on page 61.
- Connect the ByteBlaster Parallel Download Cable to your PC by following the instructions that begin on page 65.

Installing the LP6 Logic Programmer Card

Follow these steps to install the LP6 card in Pentium- and 486-based PCs:

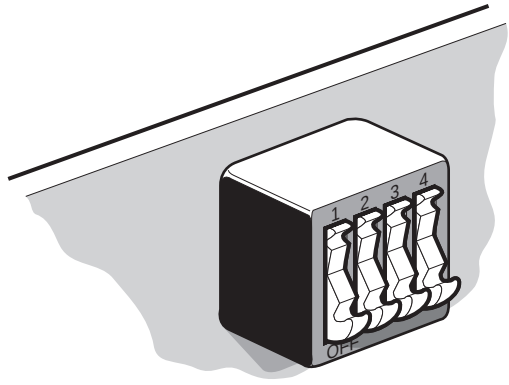
1. Be sure the computer's power is turned off.

2. Remove the cover of your computer. Refer to the documentation accompanying the computer for instructions.

3. Ensure that all four dipswitches on the LP6 card are turned on, as shown in Figure 1-4.

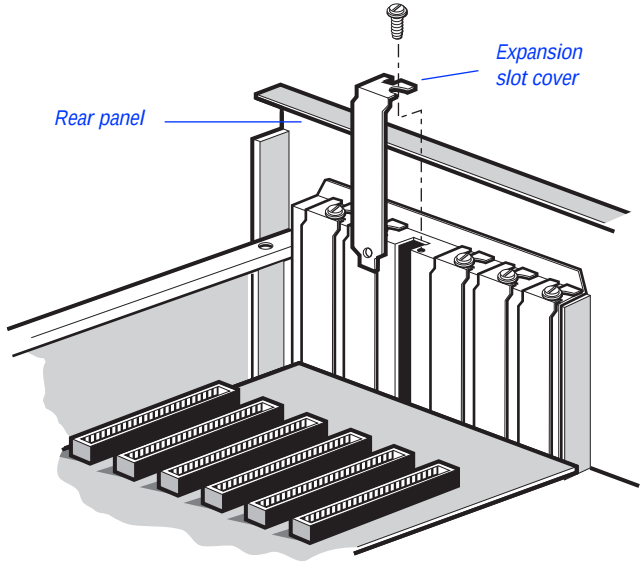
The default I/O address of the LP6 card is 280 hex. If you must change this address because of an addressing conflict, refer to "Changing the LP6 Card Address Location" on page 56 for dipswitch settings for other I/O addresses.

Figure 1-4. Default Switch Settings on the LP6 Card



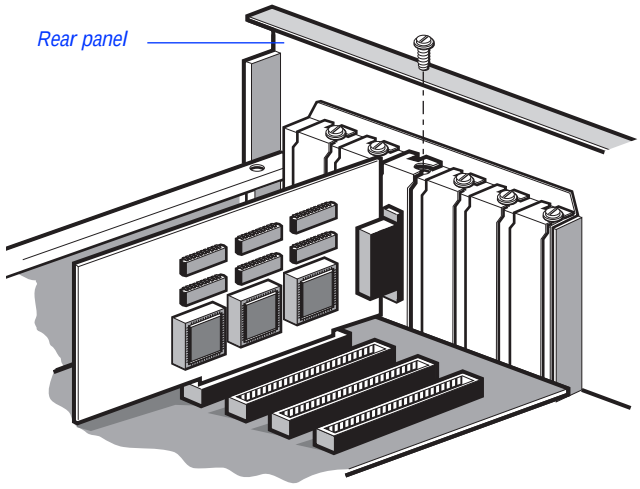
4. Select any convenient empty expansion slot for the LP6 card. If the expansion slot is covered, remove the screw that holds the expansion slot cover and remove the cover. See Figure 1-5.

Figure 1-5. Removing the Expansion Slot Cover



5. Insert the card into the expansion slot and fasten the retaining bracket with the screw from the slot cover. See Figure 1-6.

Figure 1-6. Locking the Board in Place



6. Tighten all the locking screws on all connectors.

Changing the LP6 Card Address Location

You can configure the LP6 card for different I/O addresses. Table 1-4 lists the available I/O addresses and corresponding dipswitch settings. If you use an address other than 280 hex, you must update your hardware setup in MAX+PLUS II. Go to "Changing the Hardware Setup" in MAX+PLUS II Help for instructions.

Table 1-4. LP6 Card I/O Addresses

Base I/O Address (hex)	Dipswitch Setting			
Required Address Space 16 locations	(1=ON; 0=OFF) 1	2	3	4
270	0	0	0	0
260	1	0	0	0
250	0	1	0	0
240	1	1	0	0
230	0	0	1	0
220	1	0	1	0
210	0	1	1	0
200	1	1	1	0
2F0	0	0	0	1
2E0	1	0	0	1
2D0	0	1	0	1
2C0	1	1	0	1
2B0	0	0	1	1
2A0	1	0	1	1
290	0	1	1	1
280 (default)	1	1	1	1

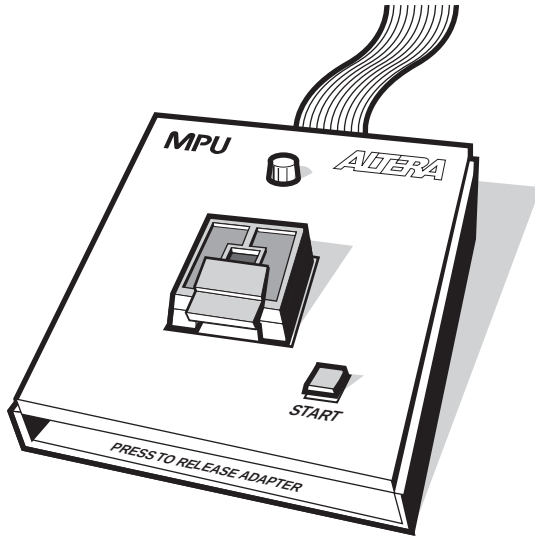
Installing the Master Programming Unit

The Master Programming Unit (MPU) consists of a base unit and one or more adapters that program MAX+PLUS II-supported Altera devices. The PL-MPU programming unit is shipped with all PC-based development systems that include programming hardware.

MAX+PLUS II also supports an older programming unit, the PLE3-12A. The PLE3-12A programs some devices from the Classic and MAX 5000 device families.

Figure 1-7 shows an assembled PL-MPU Master Programming Unit.

Figure 1-7. Master Programming Unit



To install the MPU, follow these steps:

1. On the rear panel of your PC, connect the 25-pin flat ribbon cable of the PL-MPU base unit to the Logic Programmer card.

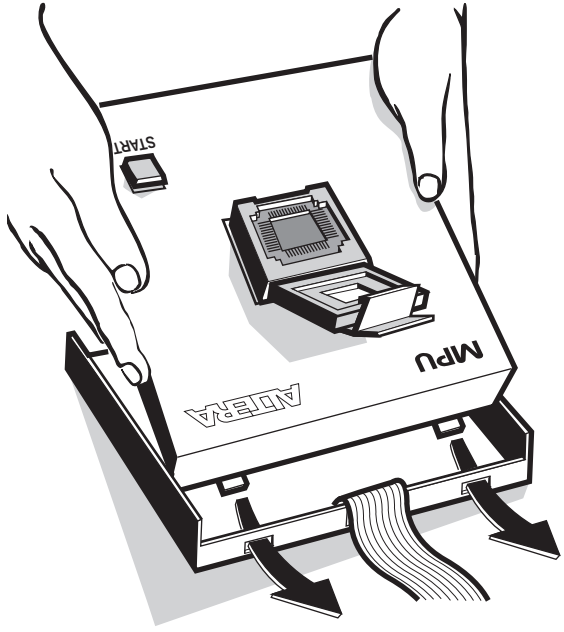
   ☞ Do *not* plug the PL-MPU ribbon cable into the parallel printer port.

2. Install the adapter by sliding the two tabs at the top of the adapter into the slots provided on the base unit. Be sure to use the appropriate adapter for the device you want to program. See Figure 1-8.

3. Carefully lower and align the other end of the adapter, so that the connector in the adapter is inserted into the socket on the base unit. Press down firmly.

4. Open the MAX+PLUS II Programmer or Simulator. Choose the **Hardware Setup** command (Options menu), and select *LP6 + PL-MPU* in the *Hardware Type* drop-down list box. Choose **OK**.

Figure 1-8. Installing the Adapter

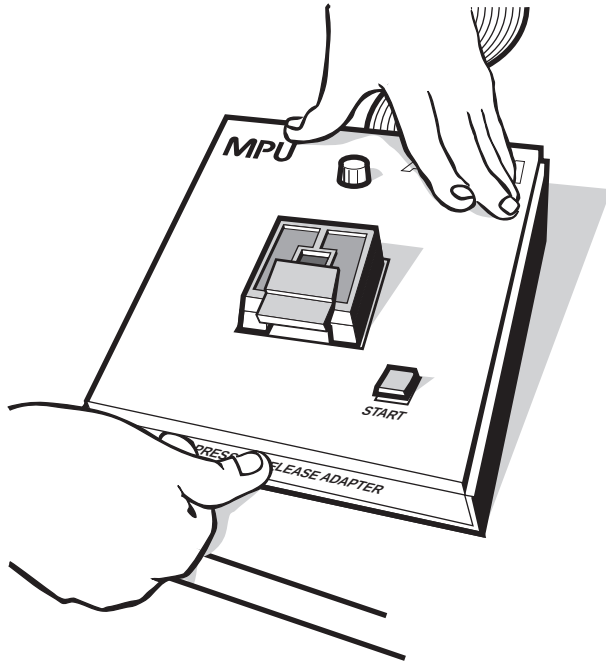


To release an adapter from the base unit:

1. Press down on the front of the unit, while holding down the other end. See Figure 1-9.

2. Lift out the adapter.

Figure 1-9. Releasing the Adapter



Installing the FLEX Download Cable on a PC

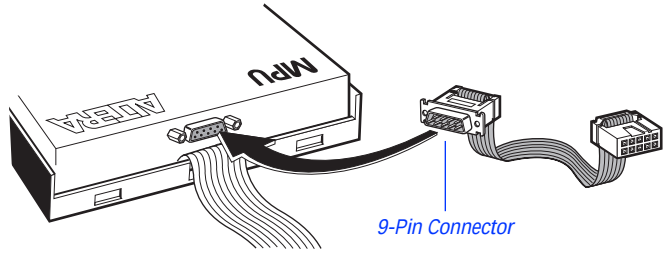
You can install an optional cable that allows you to download configuration data to configure FLEX 6000, FLEX 8000, and FLEX 10K devices in-system.

To set up the FLEX Download Cable for configuration:

1. Insert a Configuration EPROM programming adapter (e.g., PLMJ1213 or PLMT1064) into the PL-MPU base unit.

2. Connect the FLEX Download Cable to the 9-pin D-type connector on the Configuration EPROM programming adapter, as shown in Figure 1-10.

Figure 1-10. Connecting the FLEX Download Cable



3. Connect the other end of the FLEX Download Cable to the 10-pin male header on the target printed circuit board.

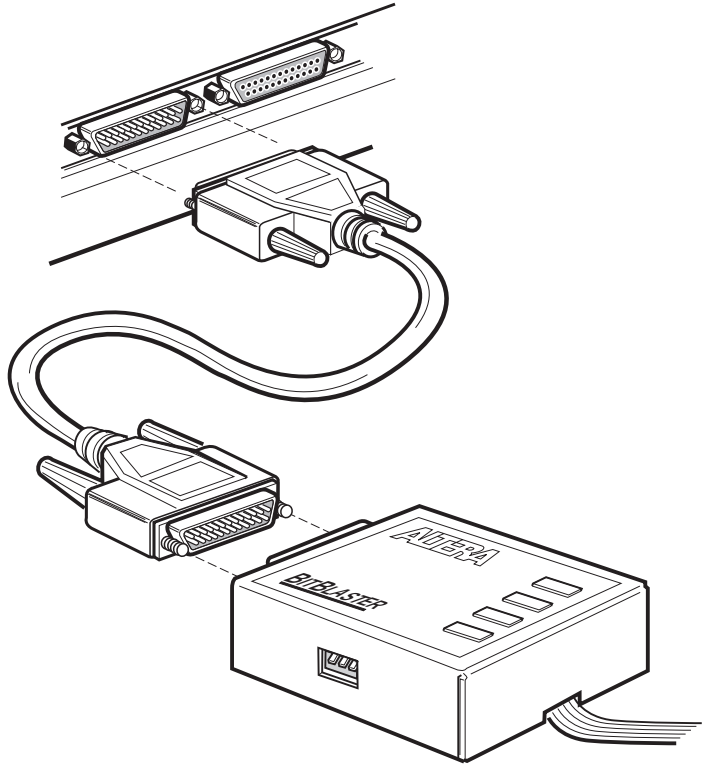
Go to *Application Note 87 (Configuring FLEX 6000 Devices)*, *Application Note 33 (Configuring FLEX 8000 Devices)*, *Application Note 38 (Configuring Multiple FLEX 8000 Devices)*, and *Application Note 59 (Configuring FLEX 10K Devices)* for instructions on how to configure FLEX 6000, FLEX 8000, and FLEX 10K devices.

Installing the BitBlaster on a PC or UNIX Workstation

The BitBlaster can be used with a PC or UNIX workstation RS-232 serial port (called a "COM port" on a PC). To set up the BitBlaster for device configuration or programming follow these steps:

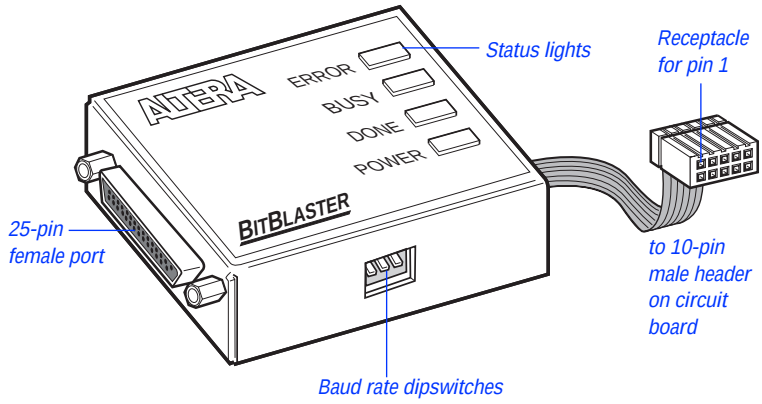
1. Connect the BitBlaster to the serial port on the computer. Figure 1-11 shows a typical serial port on a PC.

Figure 1-11. Connecting the BitBlaster to the Serial Port on the Computer



2. Connect the other end of the BitBlaster to the 10-pin male header on the target printed circuit board. See Figure 1-12.

Figure 1-12. BitBlaster and 10-Pin Female Connector



3. Open the MAX+PLUS II Programmer or Simulator. Choose the **Hardware Setup** command (Options menu), and select *BitBlaster* in the *Hardware Type* drop-down list box. Choose **OK**.

4. You must ensure that the baud rate of the BitBlaster, your computer's serial port, and the baud rate set in MAX+PLUS II are the same. If necessary, you can change the BitBlaster's baud rate by setting the dipswitches on the side panel. Dipswitch settings are listed in Table 1-5.

You should set the baud rate as high as possible to accelerate configuration time. However, some PC-based systems cannot use baud rates higher than 9600 bps.

Table 1-5. BitBlaster Baud Rate Dipswitch Settings

Baud Rate (bps)	Dipswitches 1 through 3 (1 = ON; 0 = OFF)		
9,600	1	1	1
14,400	0	1	1
19,200	1	0	1
38,400	0	0	1
57,600	1	1	0
76,800	0	1	0
115,200	1	0	0
230,400	0	0	0

Go to "Changing the Hardware Setup" in MAX+PLUS II Help for more information on setting baud rates.

Configuring an IBM RISC System/6000 Workstation Serial Port for Programming

You can use the MAX+PLUS II Programmer and the BitBlaster to program and configure devices with an IBM RISC System/6000 workstation. In order to use the Programmer, you must turn off the Carrier Detect feature on the workstation's serial port to ensure that it will not block the programming signals.

To turn off the Carrier Detect feature:

1. Become root.
2. Start the **smit** system management tool.
3. Choose the following options in order:
   a. **Devices**
   b. **TTY**
   c. **Change/Show Characteristics of a TTY**
   d. **tty0** or **tty1** (the name of the serial port you are currently using)
   e. **Change/Show TTY Program**
4. Select the Entry Field for **STTY attributes for RUN TIME**.
5. Append `,clocal` to the end of the text string in the Entry Field.
6. Select the Entry Field for **STTY attributes for LOGIN**.
7. Append `,clocal` to the end of the text string in the Entry Field.
8. Press ↵.

Installing the ByteBlaster on a PC

The ByteBlaster Parallel Download Cable is designed to be used with a PC parallel port (i.e., a printer port). You can connect the ByteBlaster directly to your PC's parallel port, or through the MAX+PLUS II Software Guard.

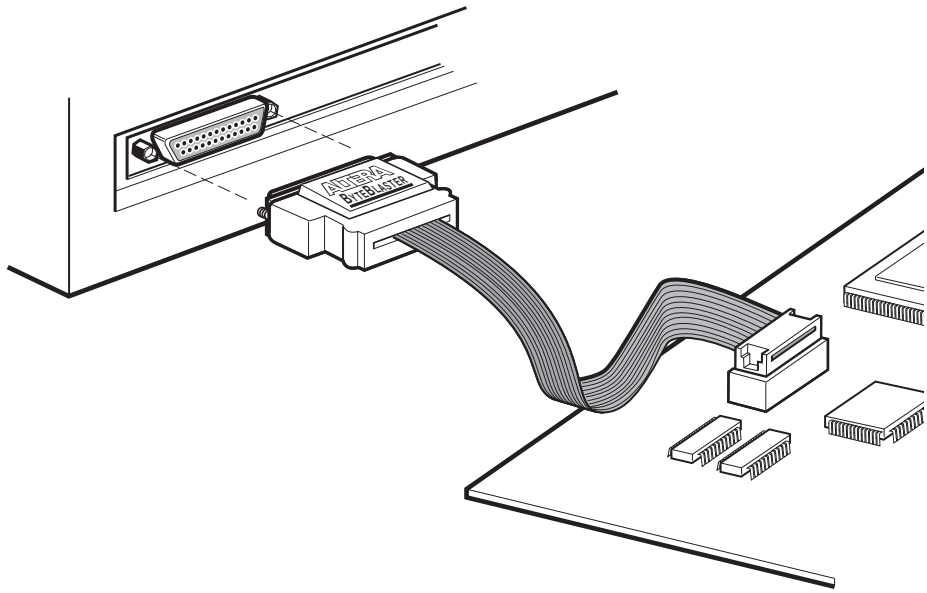
To install and set up the ByteBlaster for device configuration or programming, follow these steps:

1. If you are running the Windows NT 3.51 or 4.0 operating system, ensure that the Altera ByteBlaster driver is installed. Go to "Installing Windows NT Drivers" on page 10 for instructions.

2. Attach the ByteBlaster to a parallel port on your PC and insert the 10-pin female plug into the prototype system containing the target device, as shown in Figure 1-13.

   ☞ The board must supply power to the ByteBlaster.

Figure 1-13. ByteBlaster Parallel Port Download Cable



3. Open the MAX+PLUS II Programmer or Simulator. Choose the **Hardware Setup** command (Options menu), select *ByteBlaster* in the *Hardware Type* drop-down list box, and select the appropriate LPT port in the *Parallel Port* drop-down list box. Choose **OK**.

Go to "Changing the Hardware Setup" in MAX+PLUS II Help for more information on setting up the ByteBlaster.

Creating & Using a Local Copy of the maxplus2.ini File

The **maxplus2.ini** file is a software initialization file created during installation. It contains both Altera- and user-specified parameters that control the way MAX+PLUS II applications operate.

The **maxplus2.ini** file stores essential information for running MAX+PLUS II. For example, this file stores the pathnames of the Altera-provided primitive, megafunction, and macrofunction libraries and the default colors for your application windows.

☞ If you delete this file, you must reinstall MAX+PLUS II.

The **maxplus2.ini** file can be stored in one of several locations: for a single-user PC installation, it is stored in the MAX+PLUS II system directory (**\maxplus2**); on a UNIX workstation, it is stored in your home directory. If you use a network copy of MAX+PLUS II, or if a single computer is used by several engineers, you must create a local copy of the **maxplus2.ini** file. You must then set up an environment variable that specifies the location of the file.

To set up a local copy of the **maxplus2.ini** file on a PC:

1. Copy the existing **maxplus2.ini** file to the desired drive and directory. You can also open the file in MAX+PLUS II with the **Open** command (File menu) and save it to the desired drive and directory with the **Save As** command (File menu).

2. Specify the drive and directory of your **maxplus2.ini** file in your Windows environment:

   ✓ If you are using Windows NT 3.51 or 4.0, go through the following steps:

   a. Double-click Button 1 on the **System** icon in the Windows Control Panel. If you are using Windows NT 4.0, select the **Environment** tab.

   b. In the *Variable* box, type the following text:

      ```
      MAXPLUS2_INI
      ```

c. In the *Value* box, type the drive and directory name, e.g., `d:\maxplus2`.

d. Choose **Set**, then **OK**.

or:

✓ If you are using Windows 95, go through the following steps:

a. Edit the **autoexec.bat** file that is used to boot up your computer to include the following line:

`set MAXPLUS2_INI=`*<drive and directory name>* ↵

b. Save the changes to **autoexec.bat** and reboot your computer.

To set up a local copy of the **maxplus2.ini** file on a UNIX workstation:

✓ Copy the existing **/usr/maxplus2/maxplus2.ini** file to your home directory with UNIX workstation commands. If you need to store the **maxplus2.ini** file in a different location, perform the following additional steps:

✓ If you are running a C shell, add the line `setenv MAXPLUS2_INI <d`*irectory name>* ↵ to the **.cshrc** file in your **/usr/maxplus2** directory, then type the command `source .cshrc` ↵ at the C shell prompt.

or:

✓ If you are running a Bourne shell, add the line `set MAXPLUS2_INI=`*<directory name>* ↵ to the **.profile** file in your home directory, then type the command `source .profile` ↵ at the Bourne shell prompt.

MAX+PLUS II File Organization

During MAX+PLUS II installation, two directories are created: **\maxplus2** and **\max2work**. The **\maxplus2** directory contains system software and data files and includes the subdirectories described in Table 1-6:

The pathnames below are shown using the PC pathname convention of backslash (\) characters, but UNIX pathnames use forward slash (/) characters. On a UNIX workstation, the **max2work** and **maxplus2** directories are subdirectories of the **/usr** directory. Otherwise the file and directory organization is identical on PCs and UNIX workstations, except where noted.

Table 1-6. MAX+PLUS II System Directory Structure (Part 1 of 2)

Directory	Description
.\adm	Contains FLEXlm license manager daemon, license files, and license manager executable files (UNIX workstation and network installations only).
.\bin	Contains the executable software program files (UNIX workstation installations only).
.\common	Contains common UNIX files (UNIX workstation installations only).
.\drivers	Contains Windows NT device drivers (PC installations for Windows NT only).
.\edc	Contains Altera-provided EDIF Command Files (**.edc**) that customize EDIF Output Files (**.edo**) for specific third-party simulation environments.
.\fonts	Contains Altera, Arial, and MS Sans Serif fonts (UNIX workstation installations only).
.\hp	Contains platform-specific files for HP 9000 Series 700/800 UNIX workstation installations.
.\lmf	Contains Altera-provided Library Mapping Files (**.lmf**) that map third-party logic functions to equivalent MAX+PLUS II logic functions.
.\max2inc	Contains Include Files (**.inc**) with Function Prototypes for Altera-provided macrofunctions. Function Prototypes list the ports (pinstubs) for macrofunctions that can be implemented in Altera Hardware Description Language (AHDL) Text Design Files (**.tdf**).
.\max2lib\edif	Contains primitives and macrofunctions used for third-party EDIF interfaces.

Table 1-6. MAX+PLUS II System Directory Structure (Part 2 of 2)

Directory	Description
.\max2lib\mega_lpm	Contains megafunctions, including Library of Parameterized Modules (LPM) functions, and the corresponding Include Files that contain their AHDL Function Prototypes.
.\max2lib\mf	Contains old-style 74-series and custom macrofunctions.
.\max2lib\prim	Contains Altera-provided primitives.
.\rs6000	Contains platform-specific files for IBM RISC System/6000 UNIX workstation installations.
.\solaris	Contains platform-specific files for Solaris UNIX workstation installations.
.\sunos	Contains platform-specific files for SunOS UNIX workstation installations.
.\vhdl*nn*\altera *Note (1)*	Contains the **altera** library with the `maxplus2` package. This package includes all MAX+PLUS II primitives, megafunctions, and macrofunctions supported by VHDL.
.\vhdl*nn*\ieee *Note (1)*	Contains the **ieee** library of VHDL packages, including `std_logic_1164`, `std_logic_arith`, `std_logic_signed`, and `std_logic_unsigned`.
.\vhdl*nn*\std *Note (1)*	Contains the **std** library with the `standard` and `textio` packages defined in the ***IEEE Standard VHDL Language Reference Manual***.
Note: (1) ***nn*** represents "87" or "93," indicating VHDL 1987 or 1993 support.	

The **\max2work** directory contains tutorial and sample files and includes the subdirectories described in Table 1-7:

Table 1-7. MAX+PLUS II Working Directory Structure

Directory	Description
.\ahdl	Contains the sample files used to illustrate "How to Use AHDL" topics in MAX+PLUS II Help and in the ***MAX+PLUS II AHDL*** manual.
.\chiptrip	Contains all files for the **chiptrip** tutorial project described in this manual.
.\edif	Contains all sample files used to illustrate EDIF features in MAX+PLUS II Help.
.\tutorial	Contains the **read.me** file for the **chiptrip** tutorial. You should create the files for the **chiptrip** project in this subdirectory.

Table 1-7. MAX+PLUS II Working Directory Structure

Directory	Description
.\vhdl	Contains the sample files used to illustrate "How to Use VHDL" topics in MAX+PLUS II Help and in the ***MAX+PLUS II VHDL*** manual.
.\verilog	Contains the sample files used to illustrate "How to Use Verilog HDL" topics in MAX+PLUS II Help and in the ***MAX+PLUS II Verilog HDL*** manual.

Go to the Altera-provided Software Interface Guide for your third-party environment for information on the directory structure of the files installed for third-party interfaces to MAX+PLUS II.

Section

2

MAX+PLUS II — A Perspective

This section gives an overview of MAX+PLUS II and describes all MAX+PLUS II applications.

Go to MAX+PLUS II Help for complete and up-to-date information on all MAX+PLUS II topics.

MAX+PLUS II Logic Design

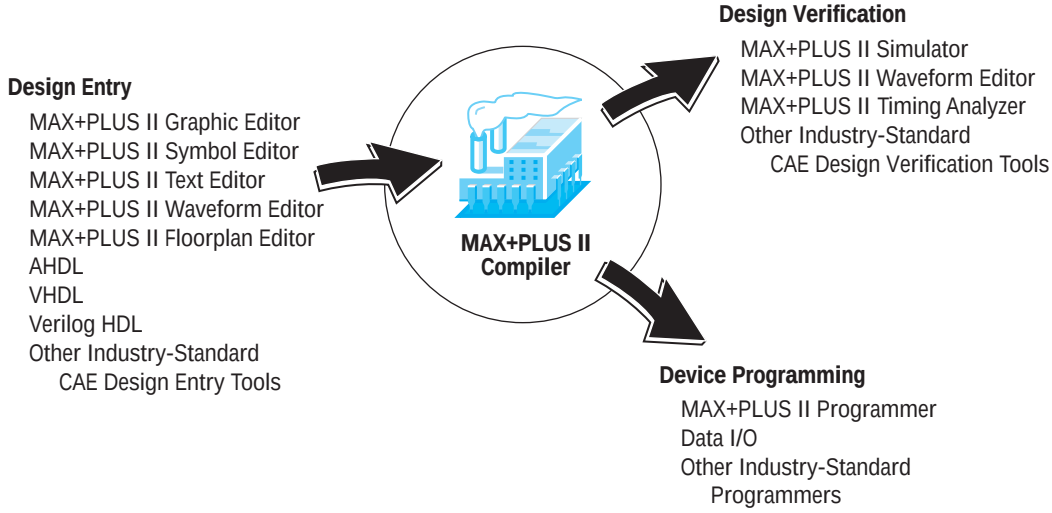
The Altera Multiple Array MatriX Programmable Logic User System (MAX+PLUS II) provides a multi-platform, architecture-independent design environment that easily adapts to your specific design needs. MAX+PLUS II offers easy design entry, quick processing, and straightforward device programming.

MAX+PLUS II development software, shown in Figure 2-1, is a fully integrated package for creating logic designs for Altera programmable logic devices—including the Classic, MAX 5000, MAX 7000, MAX 9000, FLEX 6000, FLEX 8000, and FLEX 10K families of devices.

Refer to the MAX+PLUS II **read.me** file for information on other supported Altera device families.

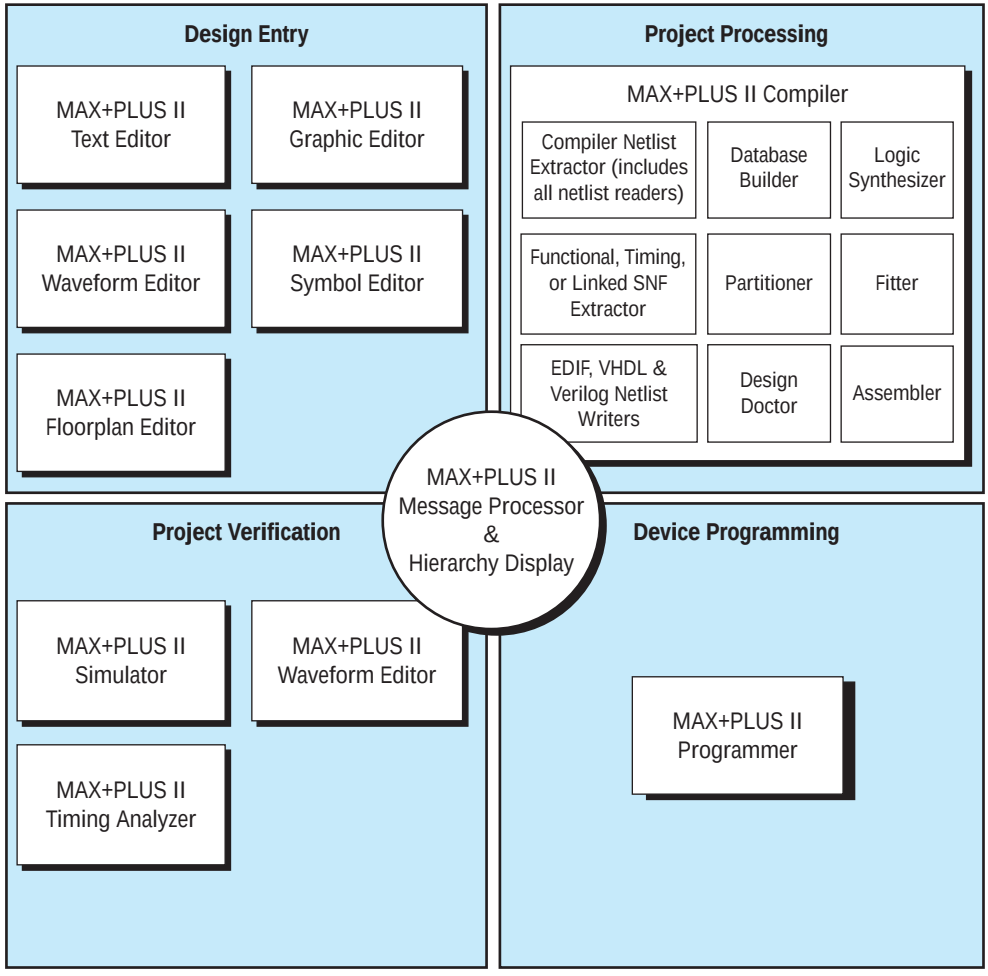
MAX+PLUS II offers a full spectrum of logic design capabilities: a variety of design entry methods for hierarchical designs, powerful logic synthesis, timing-driven compilation, partitioning, functional and timing simulation, linked multi-device simulation, timing analysis, automatic error location, and device programming and verification. MAX+PLUS II both reads *and* writes Altera Hardware Description Language (AHDL) files and standard EDIF netlist files, Verilog HDL files, VHDL files, and OrCAD schematic files. In addition, MAX+PLUS II reads Xilinx netlist files and writes Standard Delay Format (SDF) files for a convenient interface to other industry-standard CAE software.

Figure 2-1. MAX+PLUS II Design Environment



MAX+PLUS II offers a rich graphical user interface complemented with an illustrated, easy-to-use on-line help system. The complete MAX+PLUS II system includes 11 fully integrated applications that take you through every step of creating a design. (A logic design, including all subdesigns, is called a "project" in MAX+PLUS II.) See Figure 2-2.

Figure 2-2. MAX+PLUS II Applications



Many features and commands—such as opening files; entering device, pin, and logic cell assignments; and compiling the current project—are shared by many or all MAX+PLUS II applications, so that learning to use one application gives you a head start on learning to use the others. The design editors (the Graphic, Text, and Waveform Editors) and auxiliary editors (the Floorplan and Symbol Editors) also share numerous features. Each design editor allows you to perform similar tasks—such as finding a signal or symbol—in the same way. You can easily combine different types of design files in a hierarchical project, choosing the design entry format that works best for each functional block. A large library of Altera-supplied megafunctions and macrofunctions, including functions from the Library of Parameterized Modules (LPM), provide a wide range of design entry options.

You can work with different MAX+PLUS II applications simultaneously. For example, you can open multiple design files and transfer information between them while compiling or simulating another project; or, you can view an entire project hierarchy and move smoothly from one hierarchical level to another, while MAX+PLUS II automatically starts the appropriate design editor for each file.

The MAX+PLUS II Compiler lies at the heart of the MAX+PLUS II system, providing powerful project processing that you can customize to achieve the best possible silicon implementation of your project. Automatic error location and extensive documentation on error and warning messages make design modifications quick and easy. You can create output files in a variety of formats for functional, timing, and linked multi-device simulation; timing analysis; and device programming. At every step of the design process, MAX+PLUS II makes it easy for you to focus on your project—not on how to use the software.

The superb integration of the MAX+PLUS II software helps you maximize your efficiency and productivity, putting you in control of your logic design environment.

The Design Flow

The process of taking a new project from conception to completion can be simplified as follows:

1. Create a new design file or a hierarchy of multiple design files in any combination of the MAX+PLUS II design editors, i.e., the Graphic, Text, and Waveform Editors.

2. Specify the top-level design file name as the project name.

3. Assign a device family for the project. You can either allow the Compiler to select a device for you or assign a specific device.

4. Open the MAX+PLUS II Compiler window and choose the **Start** button to compile the project. If you wish, you can turn on the Timing SNF Extractor module to create a netlist file for timing simulation and timing analysis.

5. If the project compiles successfully, you can optionally perform a simulation and timing analysis:

   - To run a timing analysis, open the MAX+PLUS II Timing Analyzer window, select an analysis mode, and choose the **Start** button.
   - To run a simulation, you must first create vector inputs in a Simulator Channel File (**.scf**) in the Waveform Editor or in a Vector File (**.vec**) in the Text Editor. Then, open the MAX+PLUS II Simulator window and choose the **Start** button.

6. Open the MAX+PLUS II Programmer window and either insert a device into a programming adapter on the Master Programming Unit (MPU) or connect the BitBlaster, ByteBlaster, or FLEX Download Cable to a device that is mounted in-system.

7. Choose the **Program** button to program an EPROM- or EEPROM-based device, or choose the **Configure** button to configure an SRAM-based device.

Starting MAX+PLUS II

You can start MAX+PLUS II in one of two ways:

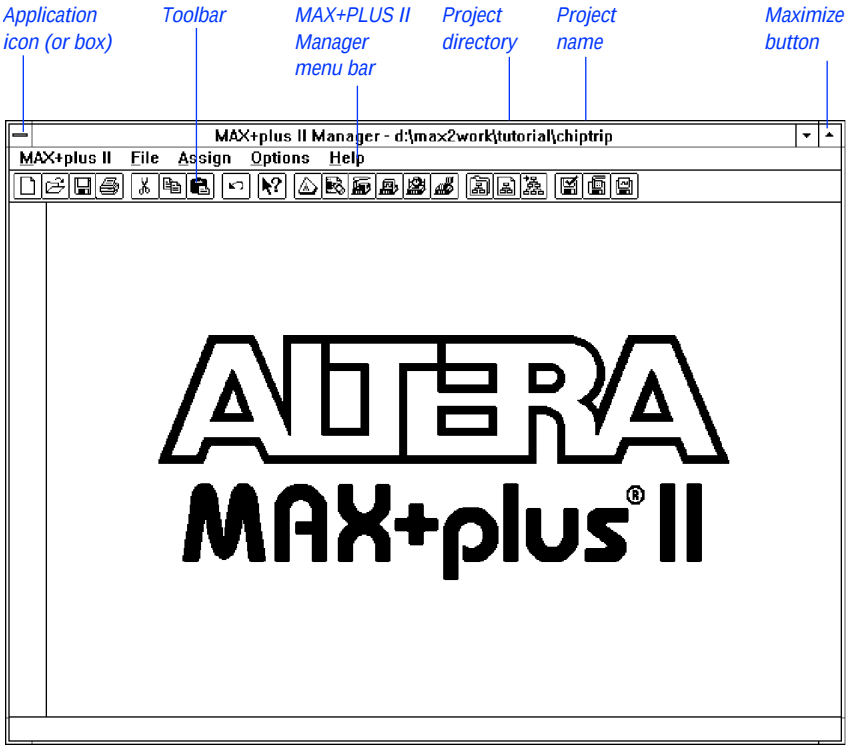
- ✓ Double-click Button 1 (the left mouse button) on the MAX+PLUS II icon. On a PC running Windows, this icon appears in the MAX+PLUS II program group.

or:

- ✓ Type `maxplus2` ↵ at the command line.

The MAX+PLUS II Manager window opens. See Figure 2-3.

Figure 2-3. MAX+PLUS II Manager Window

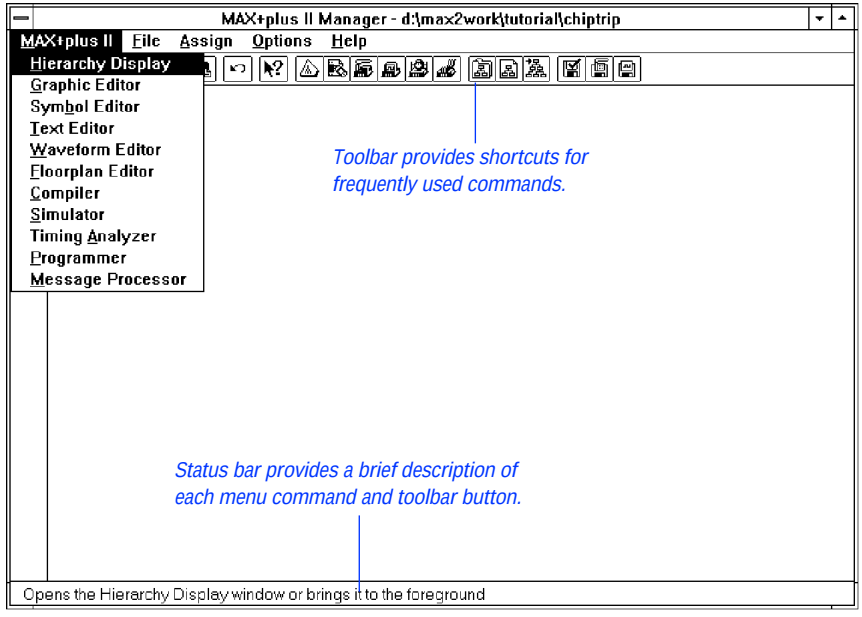


If you have not entered an authorization code or specified a network licensing file for MAX+PLUS II, the **Authorization Code** dialog box (Options menu) opens automatically. Go to "Specifying the Authorization Code for a Software Guard Installation" on page 48 or "Specifying the License File for a License File Installation" on page 49 for instructions on how to enter your authorization code or specify your network licensing file.

The MAX+PLUS II Manager

The MAX+PLUS II Manager window opens automatically when you start MAX+PLUS II. From the MAX+PLUS II menu, you can open all other MAX+PLUS II applications. See Figure 2-4.

Figure 2-4. MAX+PLUS II Menu in the MAX+PLUS II Manager Window



Commands available from MAX+PLUS II Manager menus are also available in all other MAX+PLUS II applications. For example, these common functions allow you to open a file, compile and simulate the current project, or switch to a different project. You can specify libraries of your custom symbol and design files, archive backup copies of all files in the current project in a separate directory, customize the color scheme, and enter a new authorization code. You can also show or hide the toolbar and status bar, and open MAX+PLUS II Help from the Help menu.

In addition, you can enter, edit, and delete the types of resource, device, and parameter assignments that control project compilation, including logic synthesis, partitioning, and fitting. These functions are available regardless of whether any project design file or application window is open. For information on these functions, go to "Global MAX+PLUS II Design Entry Features" on page 97.

Go to MAX+PLUS II Help for complete and up-to-date information on the MAX+PLUS II Manager.

MAX+PLUS II Applications

MAX+PLUS II software consists of 11 application programs and the MAX+PLUS II Manager. Different design entry applications can be active simultaneously, allowing you to switch between them with a click of the mouse or a menu command. At the same time, you can run one of the background applications—i.e., the Compiler, Simulator, Timing Analyzer, or Programmer. Commands shared by the various applications function in the same way, making your logic design task easier.

You can easily minimize an application window into an icon without closing the application, and restore it later. This feature allows you to keep your screen uncluttered without impairing your efficiency.

Table 2-1 describes the MAX+PLUS II applications and shows their icons.

Table 2-1. MAX+PLUS II Applications (Part 1 of 2)

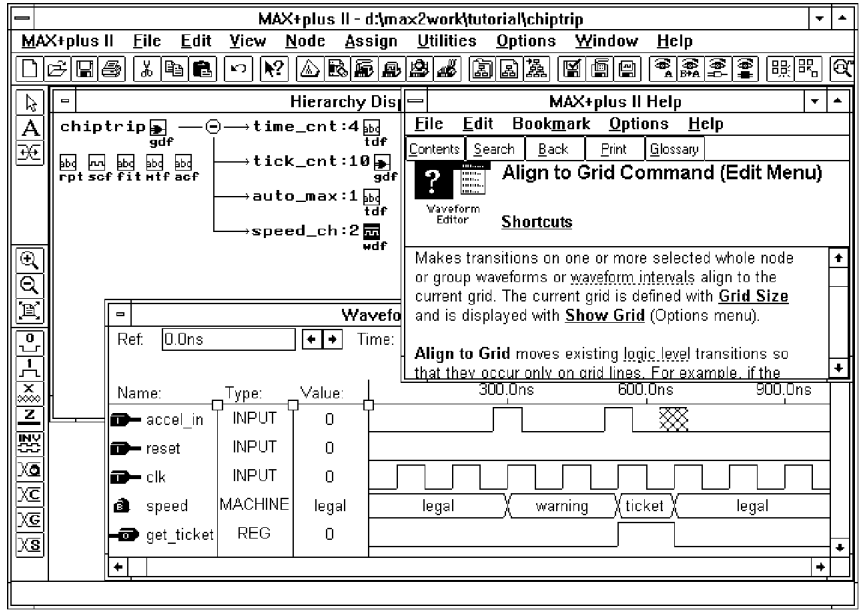
Icon	Application
	**Hierarchy Display** — Displays the current hierarchy of files as a hierarchy tree with branches that represent subdesigns. You can tell at a glance whether a design file is a schematic, text, or waveform design; which files are currently open; and which user-editable ancillary files are available for the project. You can also directly open or close one or more files in a hierarchy tree and enter resource assignments for them.
	**Graphic Editor** — Lets you enter a schematic logic design in a true what-you-see-is-what-you-get (WYSIWYG) environment. While the Altera-provided primitives, megafunctions, and macrofunctions serve as your basic building blocks, you can also use custom symbols.
	**Symbol Editor** — Allows you to edit existing symbols and create new ones.
	**Text Editor** — The Text Editor lets you create and edit text-based logic design files written in AHDL, VHDL, and Verilog HDL. With the Text Editor, you can also create, view, and edit other ASCII files used with MAX+PLUS II applications. Although you can create HDL files with other text editors, the MAX+PLUS II Text Editor allows you to take advantage of context-sensitive help, syntax coloring, and AHDL, VHDL, and Verilog HDL templates.

Table 2-1. MAX+PLUS II Applications (Part 2 of 2)

Icon	Application
	**Waveform Editor** — Serves a dual role: as a design entry tool and as a tool for entering test vectors and viewing simulation results.
	**Floorplan Editor** — Lets you assign logic to physical device pin and logic cell resources in a graphical environment. You can edit pin placements in a device package view and assign signals to individual logic cells in a more detailed Logic Array Block (LAB) view. You can also view the results of the last compilation.
	**Compiler** — Processes logic projects targeted for Altera Classic, MAX 5000, MAX 7000, MAX 9000, FLEX 6000, FLEX 8000, and FLEX 10K device families. It performs most tasks automatically. However, you can customize all or part of the compilation process.
	**Simulator** — Enables you to test the logical operation and internal timing of your logic circuit. Functional simulation, timing simulation, and linked multi-device simulation are available.
	**Timing Analyzer** — Analyzes the performance of your logic circuit after it has been synthesized and optimized by the Compiler.
	**Programmer** — Lets you program, configure, verify, examine, and test Altera devices.
	**Message Processor** — Displays error, warning, and information messages on the status of your project and allows you to locate the source of a message automatically in the original design file(s), ancillary file(s), and assignments floorplan.

Figure 2-5 shows a display of multiple windows: the Hierarchy Display and Waveform Editor windows, and a MAX+PLUS II Help topic.

Figure 2-5. Display of Multiple MAX+PLUS II Applications & Help



Design Files, Ancillary Files & Projects

Before you get started with MAX+PLUS II, you should understand the difference between design files, ancillary files, and projects.

Design Files

A *design file* is a graphic, text, or waveform file created with the MAX+PLUS II Graphic, Text, or Waveform Editor, or with another industry-standard schematic or text editor or an EDIF, VHDL, or Verilog HDL netlist writer. It contains logic for a MAX+PLUS II project and is compiled by the Compiler. The Compiler can automatically process the following design files:

- Graphic Design Files (**.gdf**)
- AHDL Text Design Files (.**tdf**)
- Waveform Design Files (**.wdf**)
- VHDL Design Files (**.vhd**)
- Verilog Design Files (**.v**)
- OrCAD Schematic Files (**.sch**)
- EDIF Input Files (.**edf**)
- Xilinx Netlist Format Files (**.xnf**)
- Altera Design Files (**.adf**)
- State Machine Files (**.smf**)

Ancillary Files

Ancillary files are files that are associated with a MAX+PLUS II project but are not part of the project hierarchy tree. Most ancillary files do not contain design logic. Some of these files are generated automatically by a MAX+PLUS II application, others are user-entered. Examples of ancillary files are Assignment & Configuration Files (**.acf**), Symbol Files (**.sym**), Report Files (**.rpt**), and Vector Files (**.vec**).

Projects

A *project* consists of all files in a design hierarchy, including ancillary input and output files. The project name is the name of the top-level design file, without the filename extension. MAX+PLUS II performs compilation, simulation, timing analysis, and programming on one project at a time, although you can always edit files belonging to another project. For example, as you compile **project1**, you may edit a TDF that is part of **project2** and save it; however, if you wish to compile it, you must first specify **project2** as the project name.

You should place each project into a separate subdirectory of the MAX+PLUS II working directory **\max2work**. (On a UNIX workstation, this directory is a subdirectory of the **/usr** directory.)

MAX+PLUS II Help

MAX+PLUS II Help provides *the* complete, up-to-date documentation on MAX+PLUS II software. Help teaches you all you need to know about each MAX+PLUS II application's basic tools, commands, procedures, shortcuts, golden rules, and messages; all primitives, megafunctions, and macrofunctions; and AHDL, VHDL, and Verilog HDL. Help also offers information on all Altera devices and adapters, allowing you to choose the appropriate device before you even begin your logic design. It points you to other Altera technical documents for additional helpful information, and provides tips on how to design most effectively with MAX+PLUS II tools.

Each Help topic contains one or more underlined words, called *jumps*, that provide links to other Help topics or to additional information on the current topic. By default, jumps are shown in green text. To view a topic, point to the jump and click Button 1 (the left mouse button) on it. A jump with a solid underline takes you to a new Help topic. A jump with a dotted underline pops up a glossary entry. A blue jump pops up an example, a list of shortcuts, or an illustration on top of the current Help topic. When you click Button 1 again, the pop-up topic closes. You can also click Button 1 on a *segmented hypergraphic*, which is a picture in a Help topic, such as a picture of a dialog box, that has links to pop-up topics.

Help is only a keystroke or a mouse click away. On-line information is accompanied by a large number of illustrations.

Go to the *MAX+PLUS II Help Poster* provided with your MAX+PLUS II system for colorful and fun explanations of how to use Help.

For information on the mechanics of using Help (e.g., copying or printing a Help topic), choose **How to Use Help** (Help menu).

The Help Menu

The menu bar of each MAX+PLUS II application provides access to the Help menu, shown in Figure 2-6.

Figure 2-6. MAX+PLUS II Help Menu

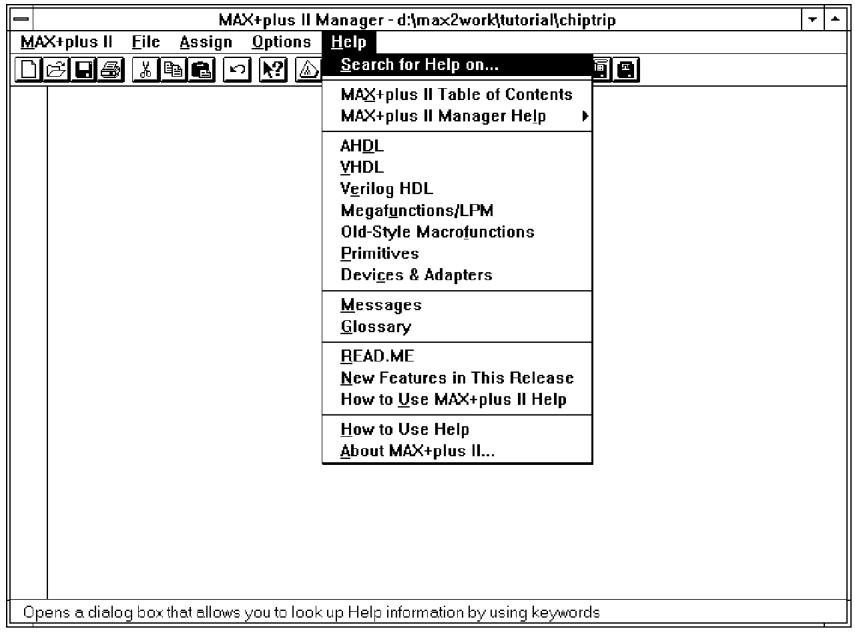


Table 2-2 describes all Help menu items and, when appropriate, shows the icons that represent them in the Help documentation.

Table 2-2. MAX+PLUS II Help Menu Items (Part 1 of 4)

Help Item	Icon
Search for Help on — Opens the **Help Topics** dialog box (called **Search** in Windows NT 3.51), which allows you to quickly search through Help's extensive index. You can select the word or phrase you want to find. If you start typing, the list box automatically scrolls to the words/phrases that most closely match what you are typing. You can then list relevant Help topics and go to a topic you wish to view.	—
MAX+PLUS II Table of Contents — A comprehensive table of contents that lists all major topics provided with MAX+PLUS II Help. It is also accessible via the **Contents** button in the button bar at the top of the Help window.	—

Table 2-2. MAX+PLUS II Help Menu Items (Part 2 of 4)

Help Item	Icon
<*Application Name*> Help — Opens a submenu of Help topics for the current MAX+PLUS II application:	*Table 2-1 shows all application icons*
Table of Contents — Table of contents for the current application.	—
Introduction — Overview of the current application, including illustrations and information on how to get started with using the application.	
Basic Tools — Detailed descriptions of items visible in an application window, as well as input and output files, accompanied by illustrations and examples. The Basic Tools Help category also provides information on primitives and macrofunctions, buttons, fields, icons, etc.	
Commands — Complete details about each command in the current application, accompanied by illustrations and examples. Illustrations of command dialog boxes include pop-up explanations of each option and button in the dialog box.	
Procedures — Step-by-step instructions, accompanied by illustrations and examples, on how to perform specific tasks in the current application.	
Golden Rules — A summary of essential tips and rules for using the current application.	
Shortcuts — Keyboard, mouse, toolbar, and tool palette shortcuts for the commands and procedures of the current application.	
AHDL — Help on AHDL, including detailed instructions on how to develop a design, and descriptions of basic elements, design structure, Backus-Naur Form (BNF) syntax, and a style guide.	
VHDL — Help on VHDL, including instructions on how to develop a design with MAX+PLUS II VHDL, and descriptions of supported VHDL constructs, Backus-Naur Form (BNF) syntax, and a style guide.	

Table 2-2. MAX+PLUS II Help Menu Items (Part 3 of 4)

Help Item	Icon
Verilog HDL — Help on Verilog HDL, including instructions on how to develop a design with MAX+PLUS II Verilog HDL, and descriptions of supported Verilog HDL constructs, Backus-Naur Form (BNF) syntax, and a style guide.	
Megafunctions/LPM — A list of megafunctions, including Library of Parameterized Modules (LPM) functions and Altera MegaCore/OpenCore functions. If you choose a specific megafunction, its function, usage rules, AHDL Function Prototype, VHDL Component Declaration, and information on device resource usage are displayed.	—
Old-Style Macrofunctions — An alphabetical list of old-style macrofunction categories. You can choose one of the categories listed to display all macrofunction names in that category. If you choose a specific macrofunction, its description, default signal logic levels, AHDL Function Prototype, VHDL Component Declaration, and function table are displayed.	—
Primitives — An alphabetical list of primitives. If you choose a specific primitive, its description, usage rules, AHDL Function Prototype, VHDL Component Declaration, and function table are all displayed.	—
Devices & Adapters — A list of all current Altera devices supported by MAX+PLUS II and their programming adapters. Selection guides for each Altera device family are also included. You can choose one of the devices listed to display a description of the device and the pin and logic cell locations for each supported device package.	
Messages — An alphabetical list of all MAX+PLUS II information, error, and warning messages. All messages are accompanied by detailed explanations of possible causes and recommended actions. Message explanations also provide jumps to additional helpful information.	
Glossary — A comprehensive list of MAX+PLUS II terms and their definitions.	
READ.ME — A copy of the **read.me** file provided with MAX+PLUS II. It gives information on system requirements, known problems, and fixes.	—

Table 2-2. MAX+PLUS II Help Menu Items (Part 4 of 4)

Help Item	Icon
New Features in This Release — A description of all new features in the current release of MAX+PLUS II, including updates to device support and interfaces to third-party EDA tools.	
How to Use Help — Information on the mechanics of using the Windows Help application.	—
How to Use MAX+PLUS II Help — Detailed information on how to use MAX+PLUS II Help, including descriptions of Help categories and documentation conventions.	
About MAX+PLUS II — Displays the MAX+PLUS II version number, current application version number, copyright and patent information, and memory and system resource usage information.	—

The Help Window Button Bar

Table 2-3 describes the buttons in the bar at the top of the Help window, which enable you to move around in Help.

Table 2-3. MAX+PLUS II Help Window Buttons

Name	Function
Contents	Shows the MAX+PLUS II Help table of contents.
Index	Opens the **Help Topics** dialog box (called **Search** in Windows NT 3.51).
Back	Goes back to previously viewed information, i.e., you retrace your path through the topics you have already viewed. The button is dimmed if there is no previous topic.
Print	Prints one or more copies of the current Help topic.
Glossary	Shows the list of MAX+PLUS II terms and their definitions. You can print any glossary entry that you open from this list.
History	Shows the last 40 topics you have viewed. You can read a topic again by double-clicking Button 1 on it. (Available only in Windows NT 3.51.)

Where to Start in Help

Help is versatile. It lets *you* decide how you want to learn about MAX+PLUS II. If you are a first-time user, however, you might find the following approach most efficient:

Step 1: From the Help menu of any application, choose the **MAX+PLUS II Table of Contents** command. This window shows all applications and other topics—including the icons that represent them—on which Help information is available.

Step 2:  From the *MAX+PLUS II Table of Contents* topic, choose *Introduction to MAX+PLUS II*. This overview summarizes the features of MAX+PLUS II and suggests starting points ("Where to Start") for becoming acquainted with MAX+PLUS II.

Step 3:  From the *Introduction to MAX+PLUS II*, you can choose *How to Use MAX+PLUS II Help* to get basic information about how Help is organized and which format conventions it uses.

Step 4:  From the Help menu in any application, choose *<application name>* **Help Procedures** to view a list of all step-by-step procedures that you can use in the current application.

Step 5:  From the Help menu in any application, choose *<application name>* **Help Golden Rules** to learn about essential tips and guidelines for the current application. Golden Rules allow you to get a head start on how to design logic circuits with MAX+PLUS II by condensing essential information that is available in other Help topics.

 Each MAX+PLUS II application provides its own *Introduction* and *Golden Rules*.

How to Request Help on a Specific Topic

MAX+PLUS II offers both menu-based and context-sensitive help. You can request help with the mouse or with the keyboard in a variety of ways:

- Every MAX+PLUS II application has a Help menu that guides you to application-specific or general MAX+PLUS II information. For example:
  - *To open the MAX+PLUS II index:* Choose **Search for Help on** from the Help menu. In the dialog box that opens, scroll to or type the desired keyword, double-click Button 1 on it to list all related Help topics, then double-click Button 1 on a topic title to open the help topic. You need not type the entire word, since the characters you type are matched with the keywords listed.
  - *To learn the shortcut for a command:* Go to the Help topic describing the command using **Search for Help On** and click Button 1 on the blue "Shortcuts" jump at the top of the topic, or go to the Help menu and choose **Shortcuts** from the <*Application Name*> Help submenu to display a table of all keyboard, mouse, toolbar, and tool palette shortcuts in the current application.
- When you choose the context-sensitive Help button () from the toolbar or press Shift+F1, the pointer turns into a question mark pointer. You can then click Button 1 on any item in the window or any menu command. If context-sensitive help is available for the item, the relevant information is displayed. Otherwise, Help shows a list of all items for which context-sensitive help is available.
- When a menu command is highlighted, or a command dialog box or a pop-up message is displayed, press F1 to get help on that topic. For example:
  - *To get instant information on a command:* Highlight the command on the menu and press F1.
  - *To see a list of all context-sensitive help items in an application:* Press F1. MAX+PLUS II Help shows a list of all items in the current application for which context-sensitive help is available. Click Button 1 on the desired item.

Design Entry

All tools necessary for creating a logic design are readily accessible in MAX+PLUS II. MAX+PLUS II accelerates your design entry with a set of standard logic functions, including primitives, megafunctions, LPM functions, and old-style 74-series-type macrofunctions. It also provides numerous basic and advanced editing features that make logic entry and debugging easier.

MAX+PLUS II provides three design entry editors—the Graphic, Text, and Waveform Editors. It also includes two auxiliary editors—the Floorplan and Symbol Editors—that facilitate design entry.

MAX+PLUS II supports a variety of design entry methods:

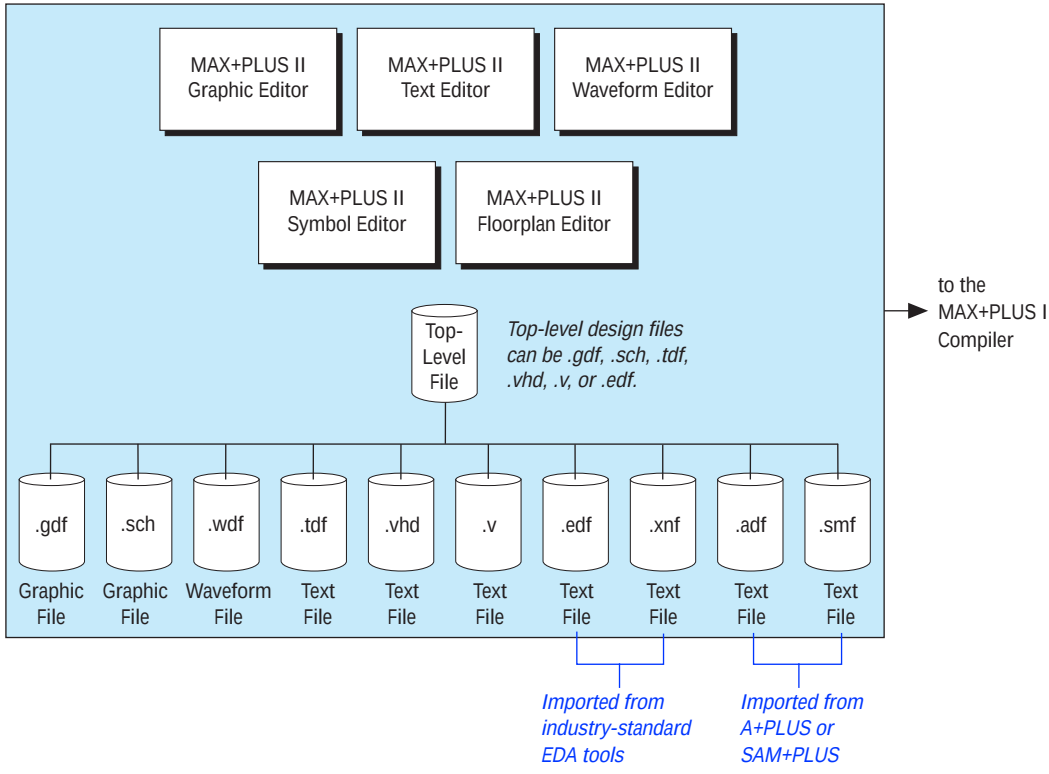
- Schematic designs are entered with the MAX+PLUS II Graphic Editor. You can also open, edit, and save schematics created with the OrCAD Draft schematic editor.
- Altera Hardware Description Language (AHDL), VHDL, and Verilog HDL designs are entered with the MAX+PLUS II Text Editor or another standard text editor.
- Waveform designs are entered with the MAX+PLUS II Waveform Editor.
- EDIF netlist files and Xilinx netlist files generated by other industry-standard EDA tools can be imported into the MAX+PLUS II environment.
- Schematic and text design files created with MAX+PLUS (DOS) and files created with Altera's A+PLUS and SAM+PLUS software packages can be integrated into the MAX+PLUS II environment.
- Physical resource assignments for any node or pin in the current project can be entered in a graphical environment with the Floorplan Editor. The Floorplan Editor saves assignments in the Assignment & Configuration File (**.acf**) for the project, which stores all types of resource, probe, and device assignments, as well as configuration settings for the Compiler, Simulator, and Timing Analyzer.
- Graphic symbols that represent any type of design file can be generated automatically in any MAX+PLUS II design editor. You can

edit the symbols or create your own custom symbols with the Symbol Editor, and use them in any schematic design file.

In a hierarchical project, you can freely mix Graphic Design Files (**.gdf**), Text Design Files (**.tdf**), VHDL Design Files (**.vhd**), Verilog Design Files (**.v**), EDIF Input Files (**.edf**), and OrCAD Schematic Files (**.sch**) at any level of the hierarchy. However, Waveform Design Files (**.wdf**), Xilinx Netlist Format Files (**.xnf**), Altera Design Files (**.adf**), and State Machine Files (**.smf**) must be either at the lowest level of a project hierarchy or be the only design file in a project. See "Project Hierarchy" on page 125.

See Figure 2-7.

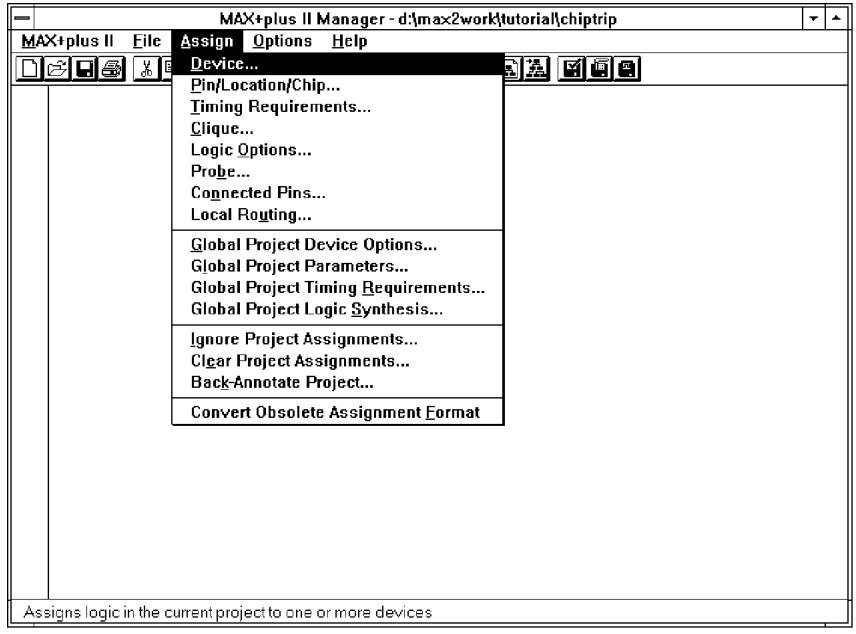
Figure 2-7. MAX+PLUS II Design Entry Methods



Global MAX+PLUS II Design Entry Features

All MAX+PLUS II applications allow you to enter, edit, and delete the types of resource, device, and parameter assignments that control project compilation—including logic synthesis, partitioning, and fitting—with commands on the Assign menu. Figure 2-8 shows the MAX+PLUS II Assign menu. You can enter assignments for the current project regardless of whether any project design file or application window is open. MAX+PLUS II saves the information in an Assignment & Configuration File (**.acf**) for the project. Assignment edits made in the Floorplan Editor window are also saved in the ACF. In addition, you can edit a project's ACF manually with the Text Editor.

Figure 2-8. MAX+PLUS II Assign Menu



The following functions are common to all MAX+PLUS II applications:

- Device, resource, and probe assignments
- Back-annotation
- Global project device options
- Global project parameters

- Global project timing requirements
- Global project logic synthesis

Device, Resource & Probe Assignments

A resource is a portion of an Altera device, such as a pin or logic cell, that performs a specific user-defined task. You can assign logic to device resources to ensure that the MAX+PLUS II Compiler fits a project exactly as you wish. The following types of assignments are available:

- *Clique assignment* Specifies which logic functions must remain together. Grouping logic functions into a clique helps ensure that they are implemented in the same Logic Array Block (LAB), Embedded Array Block (EAB), row, or device.

- *Chip assignment* Specifies which logic functions must be implemented in the same device when a project is partitioned into multiple devices.

- *Pin assignment* Assigns the input or output of a single logic function, such as a primitive or megafunction, to a specific pin, row, or column within a chip.

- *Location assignment* Assigns a single logic function, such as the output of a primitive or megafunction, to a specific location within a chip, such as a logic cell, I/O cell, embedded cell, LAB, EAB, row, or column.

- *Probe assignment* Assigns an easy-to-remember, unique name to an input or output of a logic function.

- *Connected pin assignment* Specifies how two or more pins are connected externally on your circuit board. This information is also useful for timing simulation and linked multi-project simulation.

- *Local routing assignment* Assigns a fan-out of a node to a logic cell in the same LAB as the node or an adjacent LAB to the node, using shared local interconnect. Local routing is also available between a node that is placed in an LAB on the periphery of a device and the output pin that it feeds.

- *Device assignment* Assigns project logic to a device. In a multi-device project, device assignments map chip assignments to specific devices.

You can assign a specific device and its package type, speed grade, and operating temperature. You can also assign an "AUTO" device and allow the Compiler to select a device from a target device family. This automatic device selection process can be controlled by specifying both the range and number of devices to use from the target device family. If a project is too large to fit into a specified device, you can also specify the type and number of additional devices.

- *Logic option assignment* Guides logic synthesis on individual logic functions during compilation with logic synthesis style and/or individual logic synthesis options.

  Altera provides numerous logic options, as well as three "ready-made" synthesis styles, each of which represents a collection of logic option settings combined under a single style name. You can use these styles or create your own custom styles. Synthesis styles let you tailor your synthesis options for a specific device family to take advantage of that family's architecture.

- *Timing assignment* Guides logic synthesis and fitting on individual logic functions to achieve the desired performance for input to non-registered output delays (t_{PD}), Clock to output delays (t_{CO}), Clock setup time (t_{SU}), and Clock frequency (f_{MAX}). You can also cut the connections between the timing path for a particular signal (called a "node" in MAX+PLUS II) and other nodes in the project.

Go to the current Altera ***Data Book*** and individual device data sheets for complete information on all devices.

Go to *Devices & Adapters* in MAX+PLUS II Help for pin locations for all currently available Altera device packages.

Back-Annotation

After you have entered your entire project, you can compile your project with MAX+PLUS II, then preserve, i.e., back-annotate, the resource assignments that the Compiler made during the most recent compilation so that you can produce the same fit with subsequent compilations.

Global Project Device Options

You can specify the global device options for the Compiler to use for all devices when it processes a project. To reserve additional logic capacity for future use, you can specify the percentage of pins and logic cells that must remain unused during the current compilation. You can also specify the settings for device option bits and dual-purpose device configuration pins. For example, you can specify the global default setting for the Security Bit, which prevents an EPROM- or EEPROM-based device from being interrogated.

Global Project Parameters

You can specify the names and global settings for the Compiler to use for parameters in all parameterized functions in your project.

Global Project Timing Requirements

You can enter global timing requirements for a project to specify the overall requirements for input to non-registered output delays (t_{PD}), Clock to output delays (t_{CO}), Clock setup time (t_{SU}), and Clock frequency (f_{MAX}). You can also cut the connections between all bidirectional feedback, Preset signal, and Clear signal timing paths and other timing paths in the project.

Global Project Logic Synthesis

You can select global synthesis settings for the Compiler to use when it synthesizes a project. You can specify a default logic synthesis style, specify a speed/area optimization preference, and instruct the Compiler to select automatic global control signals, such as the Clock, Clear, Preset, and Output Enable signals. You can also direct the Compiler to use standard or multi-level synthesis, one-hot state machine encoding, and automatic register packing. In addition, you can specify preferences to automatically implement logic in fast input or output logic cells and I/O cells, open-drain pins, and embedded array blocks (EABs).

Common Editor Functions

Many functions are shared by all five MAX+PLUS II editors or by the three design editors (the Graphic, Text, and Waveform Editors), making it easier for you to design logic. For example, standard functions such as saving and retrieving a file are available in all design editors.

In addition, the following functions are common to MAX+PLUS II editor applications:

Symbol & Include File Generation

You can automatically generate and update a symbol for any type of MAX+PLUS II–supported design file with the **Create Default Symbol** command (File menu). The Symbol File has the same name as the design file, with the extension **.sym**. It can be incorporated into any GDF or OrCAD Schematic File (**.sch**) that is higher up in the project hierarchy. You can also create a custom symbol for a design file.

In a process analogous to creating a default symbol, you can automatically generate and update an Include File containing an AHDL Function Prototype for any MAX+PLUS II design file. The **Create Default Include File** command (File menu) creates an Include File with the same name as the design file, with the extension **.inc**. You can use an AHDL Include Statement to incorporate the Include File and its Function Prototype into a TDF that is higher up in the project hierarchy. These Function Prototypes are required to implement instances of mega- and macrofunctions in a TDF. The MAX+PLUS II Compiler also uses the information in Include Files containing AHDL Function Prototypes to process instances of logic functions in Verilog Design Files (**.v**).

Node Location

You can select a node in an ancillary file or the current floorplan, and locate it instantly in the original design file with the **Find Node in Design File** command (Utilities menu). Similarly, you can select a node or clique in a design or ancillary file and locate it instantly in the floorplan for a project with **Find Node in Floorplan** or **Find Clique in Floorplan** (Utilities menu). The **Find Node in Floorplan** and **Find Clique in Floorplan** commands are also available in the Hierarchy Display.

Hierarchy Traversal

You can move up, down, or to the top of the current hierarchy. MAX+PLUS II opens the selected file or brings it to the front, and automatically starts the appropriate editor.

Context-Sensitive Menu Commands

Context-sensitive menu commands are available on pop-up menus in all editors and in the Hierarchy Display. These menus, which display commands that are appropriate for the selected object(s) at the current mouse pointer location, pop up automatically when you click Button 2 (the right mouse button).

Timing Analysis

You can tag nodes as sources and destinations for timing analysis, and calculate point-to-point propagation delays, setup and hold time requirements, and the maximum Clock frequency for each Clock signal in a project. You can also completely cut off a node so that only the signal path that leads to the node is included in the analysis.

Find & Replace Text

You can find and replace text—including the names of nodes and probes—in any design file in the current hierarchy.

Undo, Cut, Copy, Paste & Delete

You can undo the most recent editing step—and undo the undo. You can also copy, cut, paste, and delete one or more selected items, and paste to other MAX+PLUS II applications or Windows applications outside MAX+PLUS II.

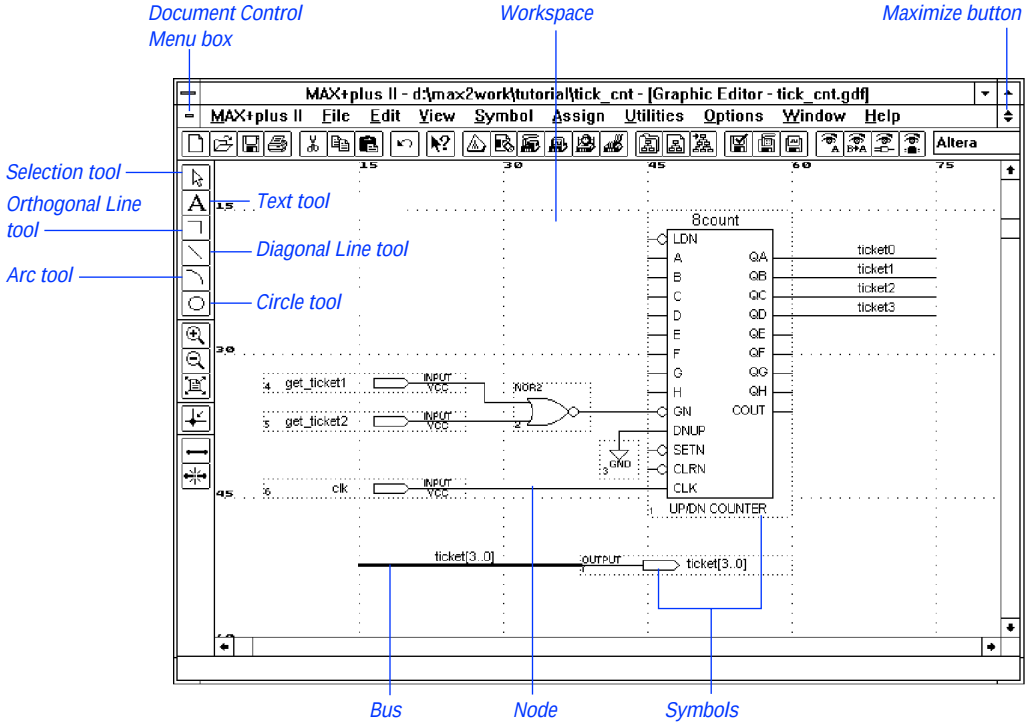
Print

You can print all or part of the current file, specify the printer or plotter, and determine the printer configuration.

MAX+PLUS II Graphic Editor

The MAX+PLUS II Graphic Editor, shown in Figure 2-9, offers a what-you-see-is-what-you-get design environment. You open a new, untitled Graphic Editor window with the **New** command (File menu), or, if no Graphic Editor window is open, by choosing **Graphic Editor** from the MAX+PLUS II menu.

Figure 2-9. MAX+PLUS II Graphic Editor



The Graphic Editor is a sophisticated schematic capture program that allows you to enter even complex designs quickly and easily. The extensive primitive, megafunction, and macrofunction libraries—including the Library of Parameterized Modules (LPM)—provide basic building blocks for constructing a design, while the symbol-generation capability lets you build your own libraries of custom functions.

A Graphic Design File (**.gdf**) or OrCAD Schematic File (**.sch**) created with the Graphic Editor can include any combination of primitive, megafunction, and macrofunction symbols. Symbols may represent any type of design file, including other GDFs and OrCAD Schematic Files, AHDL Text Design Files (**.tdf**), VHDL Design Files (**.vhd**), Verilog Design Files (**.v**), Waveform Design Files (**.wdf**), EDIF Input Files (**.edf**), Xilinx Netlist Format Files (**.xnf**), Altera Design Files (**.adf**), and State Machine Files (**.smf**).

The following features highlight the Graphic Editor's versatility:

- The "smart" Selection tool shown in Figure 2-9 makes design entry easy. This tool allows you to move and copy items and enter new symbols. When you move the Selection tool over a pinstub or the end of a line, it changes automatically to an orthogonal line-drawing tool; when you click on text, such as a pin or node name, it changes automatically into a text editing tool.

- Symbols are connected with signal lines, called nodes, or with bus lines that represent multiple logically grouped nodes. When you assign a name to a node, you can connect it to other nodes or symbols by name only. Buses are connected by name: a graphical connection is optional.

- You can customize the ports used in each separate instance of a mega- or macrofunction symbol, and optionally invert them. Any bit of a bus port can be inverted. A `NOT` "bubble" appears automatically to indicate an inverted port.

- You can select and edit multiple objects in a rectangular area or as discontiguous items. When you move a selection, the rubberbanding feature preserves signal connectivity.

- You can view probe, pin, location, chip, clique, timing, local routing, logic option, and parameter assignments on each symbol. To facilitate simulation, you can also create connected pin group assignments that specify the external device connections between pins.

- Altera-provided primitives, megafunctions, and macrofunctions reduce design entry time. You can also create your own libraries of custom functions. When you edit a symbol or regenerate a default symbol, you can automatically update selected instances or all instances of that symbol in a Graphic Editor file.

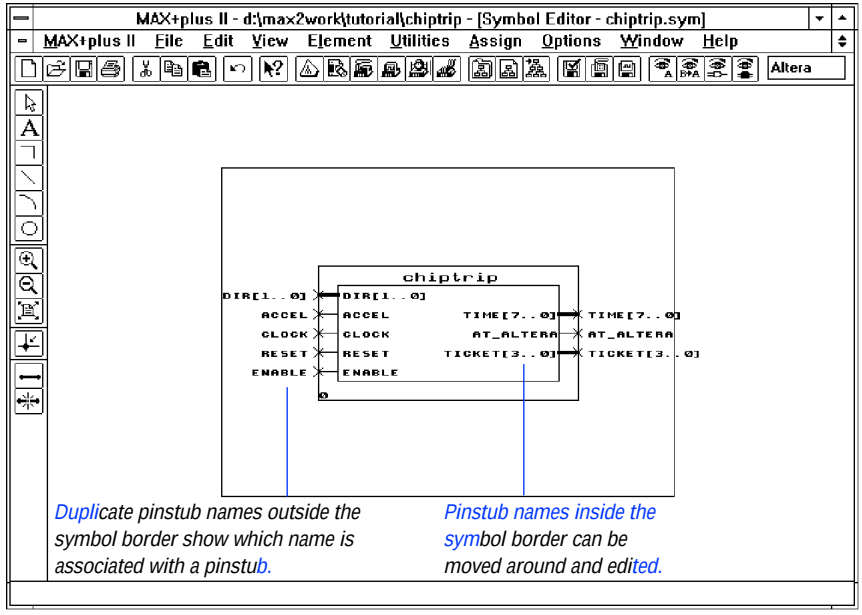
The MAX+PLUS II Graphic Editor provides many other features. For example, you can zoom in and out to various display scales so that you can see an entire design file at a glance or a detailed portion of it. You can also select various text fonts, text sizes, and line styles, and display and set the spacing of guidelines. You can copy, cut, paste, and delete one or more selected items; flip them horizontally or vertically; rotate them by 90, 180, or 270 degrees; and specify the size and horizontal or vertical orientation of the current drawing sheet.

Go to MAX+PLUS II Help for complete information on all MAX+PLUS II Graphic Editor functions and features.

MAX+PLUS II Symbol Editor

The MAX+PLUS II Symbol Editor, shown in Figure 2-10, enables you to view, create, and edit a symbol that represents a logic circuit. You open a new, untitled Symbol Editor window with the **New** command (File menu), or, if no Symbol Editor window is open, by choosing **Symbol Editor** from the MAX+PLUS II menu.

Figure 2-10. MAX+PLUS II Symbol Editor



A Symbol File has the same name as the design file it represents, with the extension **.sym**. The **Create Default Symbol** command, available from the File menu of the Graphic, Text, and Waveform Editors, creates the symbol for any design file.

The following list highlights MAX+PLUS II Symbol Editor features:

- You can customize the symbol that represents a design file.
- You can enter and edit pinstubs and pinstub names for input, output, and bidirectional pins, and specify whether pinstub names should be displayed when the symbol is entered in a Graphic Editor file.

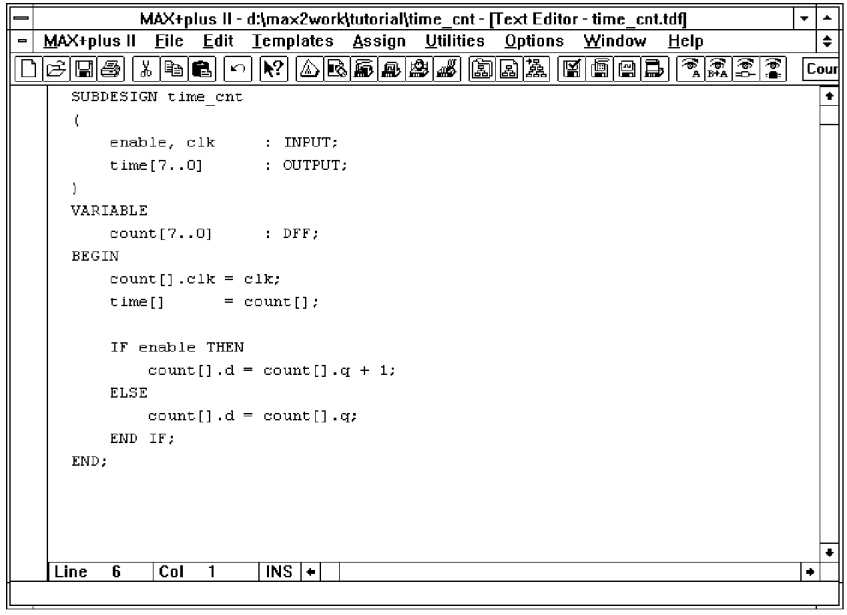
  You can choose to display the full pinstub name in a symbol or change the visible pinstub name, for example, to make it more compact or informative. Therefore, the full port name and the name displayed in a Graphic Editor file can be different.
- Pinstub names are automatically duplicated outside a symbol boundary to give you a visual reference of which name and pinstub go together. When you move a pinstub name inside the symbol boundary, the identical name outside the boundary, which cannot be moved, helps you keep track of its pinstub connection.
- You can specify parameters and their optional default values.
- The grid and guidelines help you to align objects precisely.
- You can insert comments or helpful notes in a symbol. They will also appear when the symbol is entered in a Graphic Editor file.

Go to MAX+PLUS II Help for complete information on all MAX+PLUS II Symbol Editor functions and features.

MAX+PLUS II Text Editor

The MAX+PLUS II Text Editor, shown in Figure 2-11, is a flexible tool for entering Text Design Files (**.tdf**) in the Altera Hardware Description Language (AHDL), VHDL Design Files (**.vhd**) in the Very High Speed Integrated Circuit (VHSIC) Hardware Description Language (VHDL), and Verilog Design Files (**.v**) in the Verilog HDL. You can also view, enter, and edit any other ASCII file with the MAX+PLUS II Text Editor. You open a new, untitled Text Editor window with the **New** command (File menu), or, if no Text Editor window is open, by choosing **Text Editor** from the MAX+PLUS II menu.

Figure 2-11. MAX+PLUS II Text Editor



Although you can use any ASCII text editor to create AHDL, VHDL, and Verilog HDL design files, only the MAX+PLUS II Text Editor gives you the advantage of the unique design entry, compilation, and debugging features available in MAX+PLUS II.

The following list highlights MAX+PLUS II Text Editor features:

- Because AHDL, VHDL, Verilog HDL, and the Text Editor are completely integrated into the MAX+PLUS II system, you can process an AHDL, VHDL, or Verilog HDL file with the Compiler, and the Message Processor automatically locates any syntax errors in the Text Editor. The Text Editor also provides templates for AHDL, VHDL, and Verilog HDL language constructs. (See "Altera Hardware Description Language" on page 117, "VHDL" on page 119, and "Verilog HDL" on page 121.)
- You can turn on the syntax coloring feature to allow you to clearly view language syntax in AHDL, VHDL, Verilog HDL, and a variety of text-based ancillary files.
- You can automatically find the matching section or comment delimiter for a selected delimiter, allowing you to easily move around your design file.
- You can use the drag-and-drop editing feature to move selected text to a new location within the file.
- You can manually edit Assignment & Configuration Files (**.acf**) that specify probe, resource, and device assignments, as well as project configuration settings for the Compiler, Simulator, and Timing Analyzer.
- You can create Vector Files (**.vec**) that are used as the input for simulation, functional testing, or waveform design entry. You can also create Command Files (**.cmd**) for use with the MAX+PLUS II Simulator, as well as EDIF Command Files (**.edc**) and Library Mapping Files (**.lmf**) for use with the MAX+PLUS II Compiler. You can also edit any other ASCII file.
- When you run a compilation or simulation, the MAX+PLUS II Message Processor automatically locates any syntax errors in text-based ancillary files in the Text Editor.
- With context-sensitive help, you can get immediate help on AHDL syntax elements, keywords, and statements. You can also get help on all Altera-provided primitives, megafunctions, and macrofunctions in AHDL, VHDL, and Verilog HDL design files. In addition, you can get context-sensitive help on keywords and syntax elements in other text files, such as Assignment & Configuration Files, Vector Files, Command Files, and EDIF Command Files.

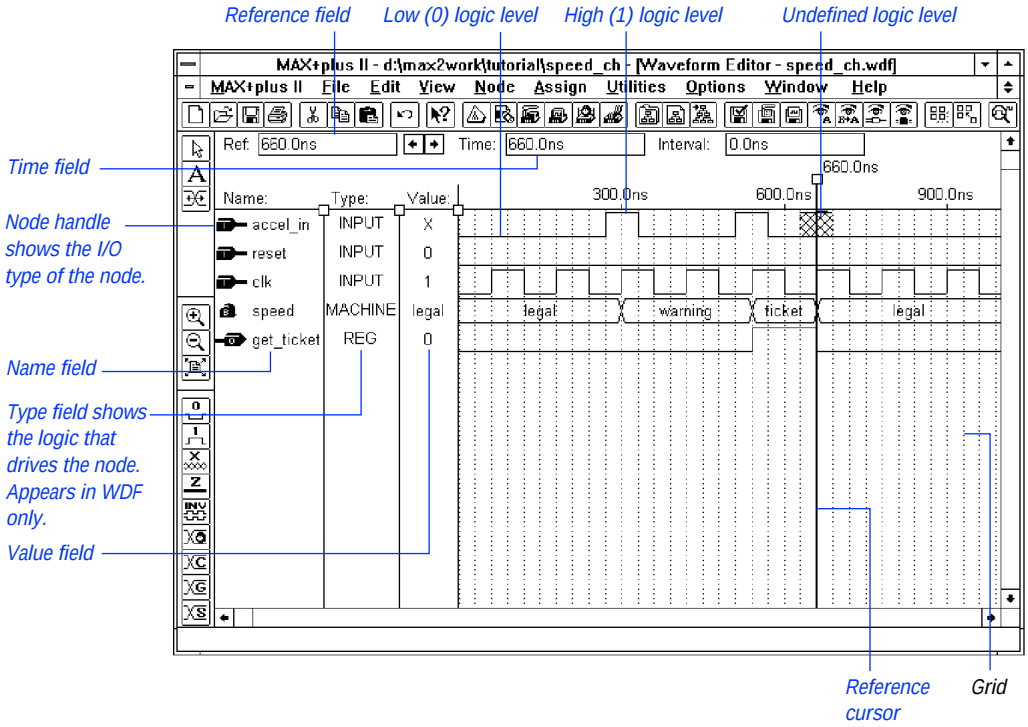
The MAX+PLUS II Text Editor provides many other features. For example, you can find, cut, copy, paste, insert, and delete text; select different text fonts and sizes; set tab stops; and use automatic indentation.

Go to MAX+PLUS II Help for complete information on all MAX+PLUS II Text Editor functions and features.

MAX+PLUS II Waveform Editor

The MAX+PLUS II Waveform Editor, shown in Figure 2-12, serves two roles: as a design entry tool and as a tool for entering test vectors and viewing simulation results. You can create Waveform Design Files (**.wdf**) that contain design logic for a project and Simulator Channel Files (**.scf**) that contain input vectors for simulation and functional testing. You open a new, untitled Waveform Editor window with the **New** command (File menu), or, if no Waveform Editor window is open, by choosing **Waveform Editor** from the MAX+PLUS II menu.

Figure 2-12. MAX+PLUS II Waveform Editor



Waveform design entry offers an alternative to graphic and text design entry. You create a waveform design by specifying combinations of input logic levels and desired outputs as graphical waveforms. A WDF can contain both logical and state machine inputs, as well as combinatorial, registered, and state machine outputs. Buried nodes can also be used to help define desired outputs.

Waveform design entry is best suited for circuits with well-defined sequential inputs and outputs, such as state machines, counters, and registers. They are ideal for combinatorial functions decoded from counters and for other repeating functions.

The Waveform Editor offers a variety of features. You can easily transform whole or partial waveforms and create and edit nodes and groups. With a few simple commands, you can create an ASCII Table File (**.tbl**) or import an ASCII Vector File (**.vec**) to create an SCF or WDF. You can also save a WDF as an SCF for simulation, or convert an SCF to a WDF for use as a design file.

The following list highlights MAX+PLUS II Waveform Editor features:

- You can create or edit a node to have an I/O type that represents an input pin, output pin, or buried logic.
- When you create a WDF, you can specify the type of logic that drives each node as pin input, registered, combinatorial, or state machine.
- You can specify a default high (1), low (0), undefined (X), or high-impedance (Z) logic level on a logic node, or any default state name on a state machine-type node.
- You can easily add any or all nodes from the Simulator Netlist File (**.snf**) for a fully optimized, compiled project to an SCF to simplify test vector creation.
- You can combine from 2 to 256 nodes to create a new group (bus), or undo grouping to expand a group into its original members. Groups can also be combined into other groups. The group value can be displayed in binary, decimal, hexadecimal, or octal radix, with or without conversion to Gray code.
- You can copy, paste, move, or delete a selected portion ("interval") of a waveform, a whole waveform, or an entire node or group (i.e., the node or group name plus waveform). With a single operation, you can edit multiple intervals, whole waveforms, and entire nodes and groups. Copies of entire nodes and groups are linked, so that waveform edits in one copy are reflected in all copies. You can also invert, insert, overwrite, repeat, expand, or compress a waveform interval of any length with any logic level, clock signal, count sequence, or state name.
- You can define and optionally display a drawing grid for aligning logic level transitions either before or after they are created.

- You can insert comments between waveforms at any point within a file.
- You can zoom in and out to any scale.
- To show the differences between simulation outputs and actual device outputs, you can superimpose any outputs on the current file, or superimpose a second Waveform Editor file to compare its node and group waveforms with those in the current file.

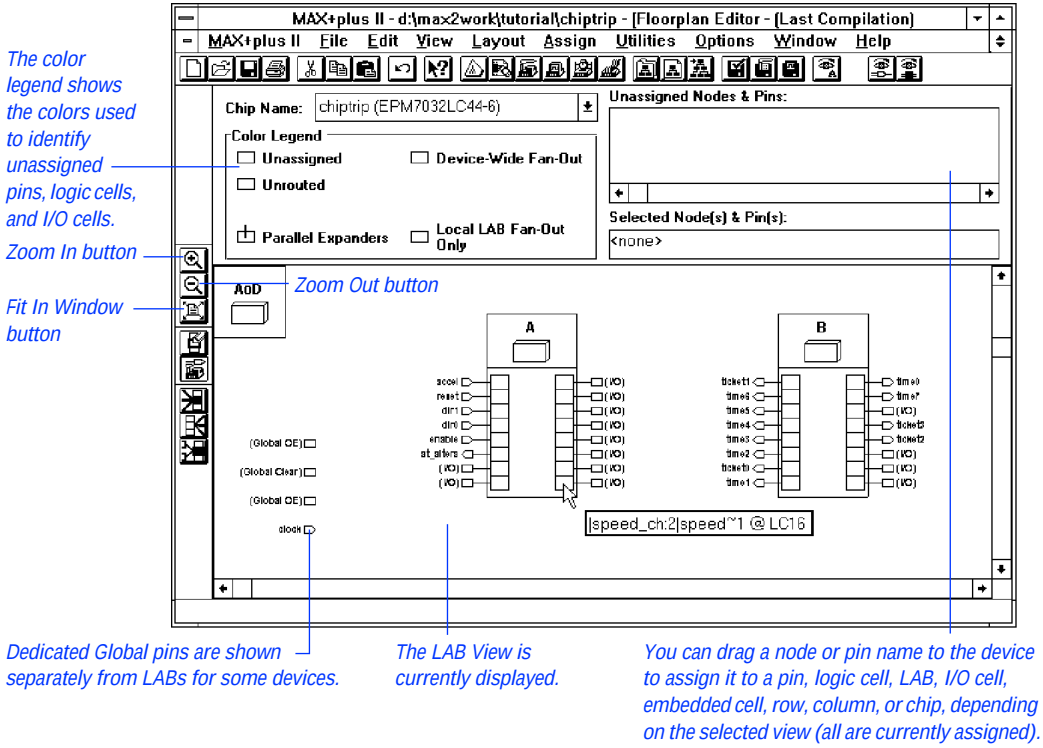
For additional information on waveform editing, go to "MAX+PLUS II Waveform Editor" under "Project Verification" on page 141.

Go to MAX+PLUS II Help for complete information on all MAX+PLUS II Waveform Editor functions and features.

MAX+PLUS II Floorplan Editor

You can use the MAX+PLUS II Floorplan Editor, shown in Figure 2-13, to assign physical device resources and to view Compiler partitioning and fitting results. You open the Floorplan Editor window by choosing **Floorplan Editor** from the MAX+PLUS II menu.

Figure 2-13. MAX+PLUS II Floorplan Editor



The Floorplan Editor provides a convenient method to enter and edit physical device resource assignments for your project. Two displays are available:

- The Device View shows all pins on a device package and their functions.
- The LAB View shows the interior of the device, including all Logic Array Blocks (LABs) and the individual logic cells within each LAB. In

devices that include Embedded Array Blocks (EABs), you can view individual embedded cells within each EAB. In MAX 9000, FLEX 6000, FLEX 8000, and FLEX 10K devices, I/O cell locations are also displayed. In addition, pins are displayed around the edges of the device packages.

The Floorplan Editor provides a list of unassigned node and pin names in your project. Each name has a handle that you can drag to an individual pin, logic cell, I/O cell, or embedded cell in the Device View or LAB View display. You can also drag a node or pin with an existing assignment back to the list of unassigned nodes or to a different location on the device.

You can also make a more general assignment to the assignment "bin" for an entire LAB, EAB, or a device, and then allow the Compiler to select the most appropriate location within the LAB or device. In MAX 9000, FLEX 6000, FLEX 8000, and FLEX 10K devices, you can also assign nodes and pins to row and column bins. Each generic assignment bin displays a number showing the number of nodes or pins assigned to it.

The following list highlights MAX+PLUS II Floorplan Editor features:

- You enter physical resource assignments in a graphical drag-and-drop environment. The Compiler's assignments can also be back-annotated and edited.
- A color legend clearly indicates unassigned and assigned pins, logic cells, and I/O cells; the type of fan-out from each item; and `VCC`, `GND`, and reserved pins.
- You can view and edit your current assignments, which are stored in the project's Assignment & Configuration File (**.acf**). You can also display a non-editable (read-only) view of the results of the last compilation, which are stored in the Fit File (**.fit**), regardless of whether fitting was successful. Any items with illegal assignments are highlighted in the list of unassigned node and pin names. Nodes that have been placed but not routed are indicated in red.
- You can automatically display the fan-in and fan-out of any selected item(s), or the paths between multiple selected items. You can also view detailed routing statistics for selected item(s) and for the most congested area of a chip.
- A Report File (**.rpt**) equation viewer allows you to select one or more items in the window and view their equations and the names of all nodes and pins that feed or are fed by any of the selected item(s). You

can select a fan-in or fan-out node and view its equation, tracing signal paths throughout the floorplan.

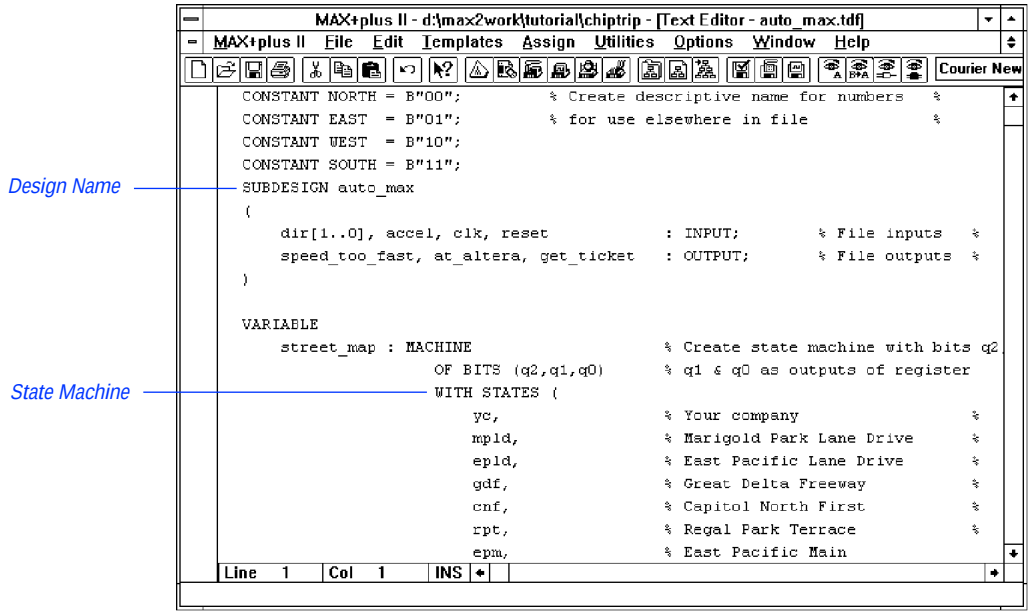
- The name of a logic function assigned to a pin or logic cell is displayed automatically in "balloon text" as the mouse pointer passes over it.
- If multiple items are assigned to a single location, you can view a list of all items and select a single item to edit.
- You can assign the same pin name to the output of one device and the input of another to control partitioning in a multi-device project.
- With the Compiler's "smart recompile" feature, you can fine-tune assignments in the Floorplan Editor and quickly recompile.

The MAX+PLUS II Floorplan Editor provides many other features. For example, you can zoom in and out to various display scales, so that you can see an entire device at a glance or a detailed portion of it. You can also copy, cut, paste, and delete one or more selected assignments, and search for an assignment by its clique name, node name, or pin, logic cell, I/O cell, or embedded cell number.

Altera Hardware Description Language

The Altera Hardware Description Language (AHDL) is a high-level, modular language that is completely integrated into the MAX+PLUS II system. You can use the MAX+PLUS II Text Editor or another text editor to create AHDL Text Design Files (**.tdf**), which you compile and simulate in MAX+PLUS II. See Figure 2-14.

Figure 2-14. AHDL Text Design File



You can use any ASCII text editor to create TDFs. However, when you enter AHDL files with the MAX+PLUS II Text Editor, you can take advantage of the unique design entry, compilation, and debugging features available only in MAX+PLUS II editors. For example, you can take advantage of AHDL templates; use syntax coloring to easily view different sections of the file; get context-sensitive help about AHDL syntax elements, keywords, and statements, as well as Altera-provided primitives, megafunctions, and macrofunctions; make resource and device assignments; and use the MAX+PLUS II automatic error location feature during and after compilation.

AHDL consists of a variety of elements and behavioral statements that describe logic. The following list highlights the features that make AHDL an ideal tool for describing functions such as state machines, truth tables, Boolean equations, conditional logic, and group operations:

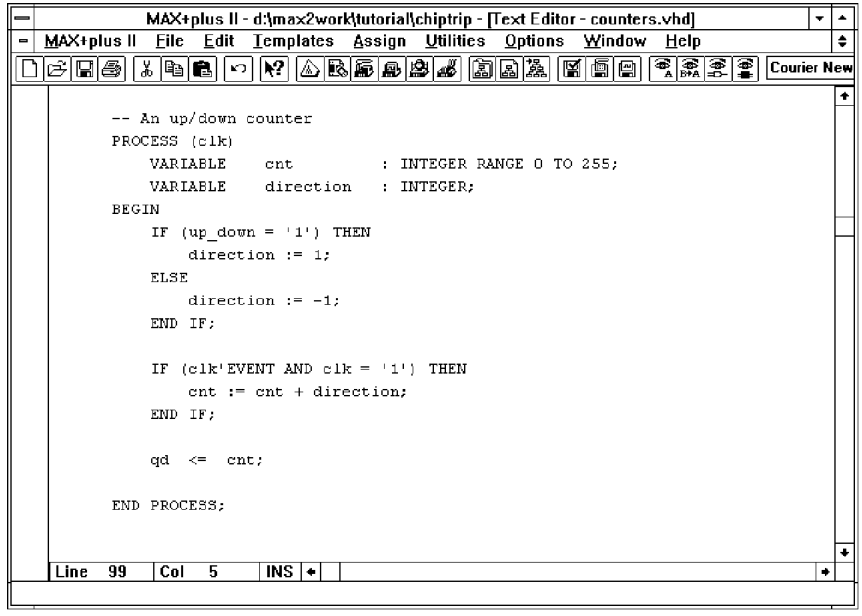
- You can use a variety of logic functions from the Library of Parameterized Modules (LPM) to implement logic. The LPM provides gate, arithmetic, and storage components that implement combinatorial logic such as decoders, multiplexers, and adders, and sequential logic such as registers and counters.
- As an alternative to using LPM functions, you can implement combinatorial and sequential logic with Boolean expressions and equations, macrofunctions, and truth tables.
- You can create your own parameterized designs in AHDL using iterative and conditional logic generation.
- You can store frequently used constants, evaluated functions, parameters, and function prototypes in Include Files (**.inc**) and incorporate them into any TDF.
- AHDL is ideal for state machine designs. The language is structured so that you can either assign state bits and state values yourself, or let the Compiler do the work for you. You can also import and export AHDL state machines between TDFs and other design files in a design hierarchy.
- The MAX+PLUS II Compiler can generate AHDL Text Design Export Files (**.tdx**) and Text Design Output Files (**.tdo**) when you compile a project. Regardless of your original design entry method, you can then rename the file as a TDF and use it to replace the original design file(s).

Go to MAX+PLUS II Help or to the ***MAX+PLUS II AHDL*** manual for complete information on AHDL.

VHDL

The Very High Speed Integrated Circuit (VHSIC) Hardware Description Language (VHDL) is a high-level, modular language that is completely integrated into the MAX+PLUS II system. VHDL is an industry-standard hardware description language that describes the inputs and outputs, behavior, and function of circuits. This language is defined by the IEEE 1076-1987 and 1076-1993 Standards. You can use the MAX+PLUS II Text Editor or another text editor to create VHDL Design Files (**.vhd**) in VHDL 1987 or 1993 syntax, which you compile and simulate in MAX+PLUS II. See Figure 2-15.

Figure 2-15. VHDL Design File



You can use any ASCII text editor to create VHDL Design Files (**.vhd**). However, when you enter VHDL Design Files with the MAX+PLUS II Text Editor, you can take advantage of the unique design entry, compilation, and debugging features available only in MAX+PLUS II editors. For example, you can take advantage of VHDL templates; use MAX+PLUS II context-sensitive help to learn about Altera-provided primitives, megafunctions, and macrofunctions; use syntax coloring to easily view different sections of the file; make resource and device assignments; and use the MAX+PLUS II automatic error location feature during and after compilation.

VHDL Design Files can contain any combination of MAX+PLUS II-supported constructs. They can also contain Altera-provided primitives, megafunctions, and macrofunctions, i.e., lower-level design files, as well as user-defined mega- and macrofunctions.

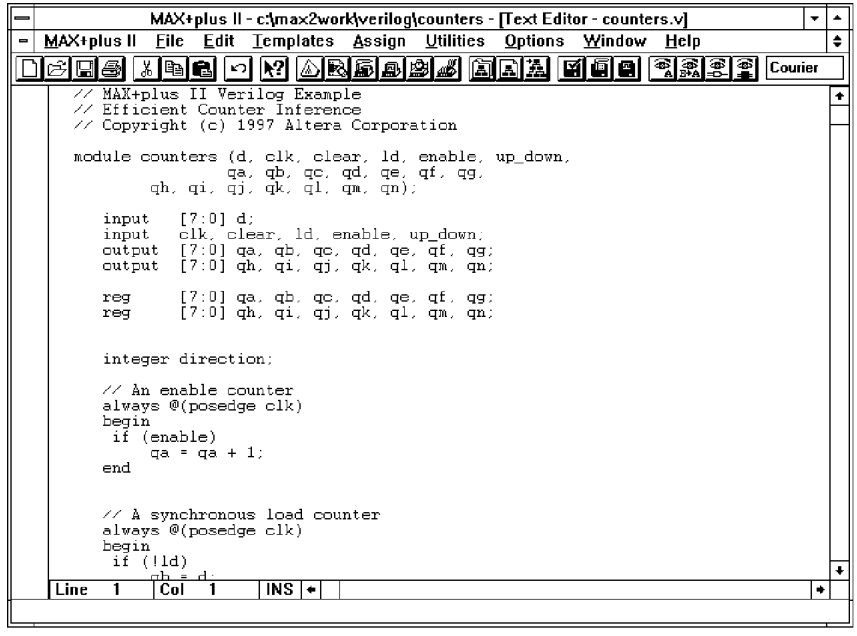
The MAX+PLUS II Compiler can generate VHDL Output Files (**.vho**) containing a project's post-synthesis functional and timing information. These files can be exported to an industry-standard simulator for simulation. Timing information can also be written to Standard Delay Format (SDF) Output Files (**.sdo**).

Go to MAX+PLUS II Help or to the ***MAX+PLUS II VHDL*** manual for information on MAX+PLUS II VHDL support.

Verilog HDL

The Verilog Hardware Description Language (HDL) is a high-level, modular language that is completely integrated into the MAX+PLUS II system. Verilog HDL is an industry-standard hardware description language that describes the inputs and outputs, behavior, and function of circuits. This language is defined by the IEEE Std 1364. You can use the MAX+PLUS II Text Editor or another text editor to create Verilog Design Files (**.v**), which you compile and simulate in MAX+PLUS II.
See Figure 2-16.

Figure 2-16. Verilog Design File



You can use any ASCII text editor to create Verilog Design Files (**.v**). However, when you enter Verilog Design Files with the MAX+PLUS II Text Editor, you can take advantage of the unique design entry, compilation, and debugging features available only in MAX+PLUS II editors. For example, you can take advantage of Verilog HDL templates; use MAX+PLUS II

context-sensitive help to learn about Altera-provided primitives, megafunctions, and macrofunctions; use syntax coloring to easily view different sections of the file; make resource and device assignments; and use the MAX+PLUS II automatic error location feature during and after compilation.

Verilog Design Files can contain any combination of MAX+PLUS II-supported constructs. They can also contain Altera-provided logic functions—including primitives, megafunctions, and macrofunctions—as well as user-defined logic functions.

The MAX+PLUS II Compiler can generate Verilog Output Files (**.vo**) containing a project's post-synthesis functional and timing information. These files can be exported to an industry-standard simulator for simulation. Timing information can also be written to Standard Delay Format (SDF) Output Files (**.sdo**).

Go to MAX+PLUS II Help or to the ***MAX+PLUS II Verilog HDL*** manual for information on MAX+PLUS II Verilog HDL support.

Primitives, Megafunctions, & Macrofunctions

Altera provides libraries of logic functions—primitives, megafunctions, and old-style (e.g., 74-series) macrofunctions—including functions that are optimized for the architecture of a particular device family. During installation, all logic functions are copied to subdirectories of the **\maxplus2\max2lib** and **\maxplus2\vhdl*nn*** directories, where ***nn*** is "87" or "93." (On a UNIX workstation, the **maxplus2** directory is a subdirectory of the **/usr** directory.)

MAX+PLUS II Help provides extensive information on all Altera-provided logic functions, including the default input logic levels, an AHDL Function Prototype, VHDL Component Declaration, and a function table. You simply choose the context-sensitive Help button on the toolbar or press Shift+F1 in the Graphic or Text Editor, then click Button 1 on a logic function symbol or logic function name to open the appropriate Help topic.

Primitives

Primitives—buffer, flipflop, latch, input/output, and logic primitives—are basic functional blocks used to design circuits with MAX+PLUS II. They can be used in GDFs, and in AHDL, VHDL, and Verilog HDL design files.

Primitives in HDL design files are a subset of the primitive symbols used in GDFs. Other primitive functions can be represented by logical operators, ports, and various statements. AHDL Function Prototypes for primitives are built into the MAX+PLUS II software. VHDL Component Declarations for primitives are provided in the `maxplus2` package in the **altera** library.

In a Graphic Editor schematic, you can create a primitive array, in which a single primitive connected to one or more named bus lines represents a series of identical primitives. During project processing, the Compiler automatically translates a primitive array into the correct number of individual primitives. Primitive arrays provide an alternative to using parameterized functions.

Megafunctions

Megafunctions are complex or high-level building blocks that can be used together with primitives and other mega- and macrofunctions to create a logic design.

Many megafunctions, including functions from the Library of Parameterized Modules (LPM), are inherently parameterized for size, behavior, and silicon implementation. The scalability of LPM and other parameterized functions can greatly simplify design entry. Megafunctions can be used freely in GDFs and in all HDL design files. When the Compiler analyzes the complete logic circuit, it automatically uses any available device-family-specific megafunction logic, and removes all unused gates and flipflops to ensure optimum design efficiency.

Old-Style Macrofunctions

Old-style macrofunctions are high-level building blocks that can be used together with primitives and mega- and macrofunctions to create a logic design. They can be used freely in GDFs and in all HDL design files. When the Compiler analyzes the complete logic circuit, it automatically uses any available device-family-specific macrofunction logic, and removes all unused gates and flipflops to ensure optimum design efficiency. All macrofunction inputs also have default input signal levels so that unused pins can be left unconnected.

Many macrofunctions have bus equivalents, which are functionally identical to the macrofunction, but have input and output pins that are grouped into buses.

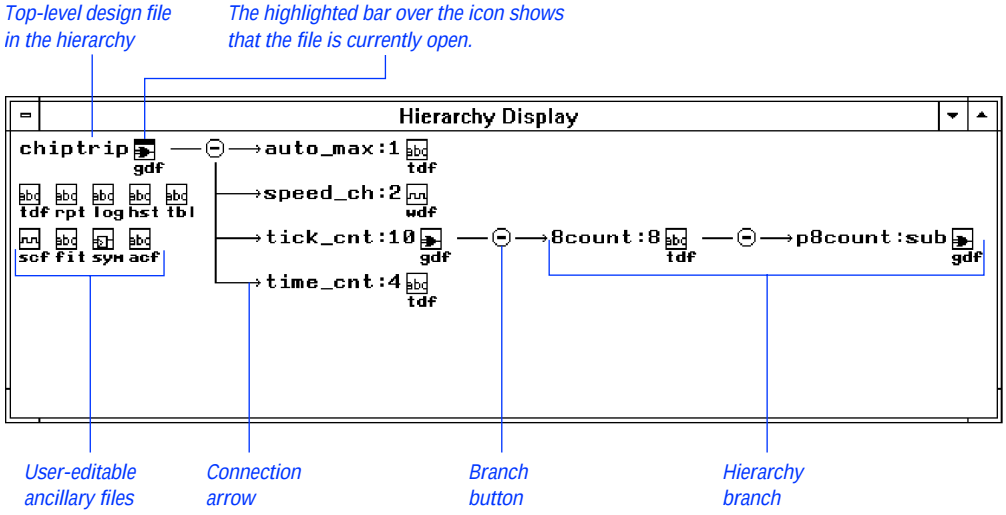
Old-style macrofunctions are not inherently parameterized. However, some Altera-specific parameters can be applied to macrofunctions to determine their style of implementation.

Altera recommends using LPM megafunctions rather than equivalent old-style macrofunctions. LPM and other parameterized functions are easier to use, scalable, and are implemented more efficiently in silicon.

Project Hierarchy

The MAX+PLUS II Hierarchy Display shows a hierarchical logic design as a hierarchy tree where lower-level design files are represented as branches. Different design entry methods can be mixed in a single project. See Figure 2-17.

Figure 2-17. MAX+PLUS II Hierarchy Display



When you open the Hierarchy Display, it shows the full hierarchy of design files—called a "hierarchy tree"—for the current project or another hierarchy of design files. If one or more files in the hierarchy are open, the top of its file icon displays a highlighted bar. The Hierarchy Display shows the entire hierarchy of design files, as well as all user-editable ancillary files for the top-level design file, if the project has been compiled with the Compiler Netlist Extractor module.

The Hierarchy Display features make it easy for you to move between the different types of files for a project. For example, you can open and close one or more files in the Hierarchy Display window; the appropriate editors are then automatically opened or closed. You can also zoom in and out to various display scales to see all or part of the hierarchy, or choose a compact display to view as many branches as possible of a large hierarchy tree.

In addition, the Hierarchy Display offers the following features:

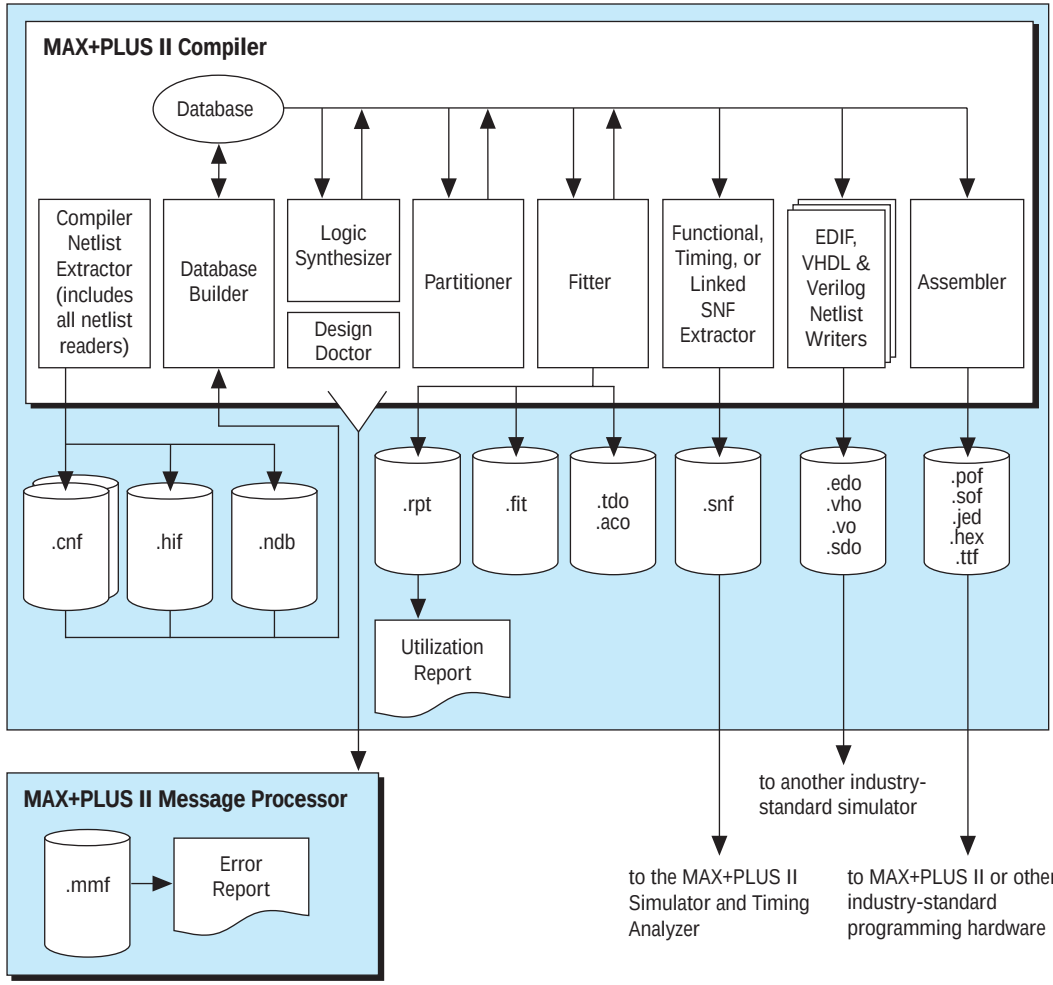
- You can easily open an editor window onto any design or ancillary file in the current hierarchy.
- Branch buttons at the intersections between hierarchy tree branches allow you to hide or display the lower-level branches.
- All filenames in the hierarchy tree are accompanied by the appropriate MAX+PLUS II editor icon and filename extension. The top-level file also shows icons and filename extensions for one or more ancillary files.
- When a file is open, a highlighted bar is displayed over the file icon. The highlighting disappears when you close the file.
- You can select a design file and view its physical implementation in the LAB View of the Floorplan Editor.
- You can select a design file and enter resource assignments for the entire file. This behavior is analogous to entering assignments on a symbol in a Graphic Editor file.
- You can display any one of multiple open hierarchies.
- You can display the hierarchy in horizontal or vertical orientation.
- You can print the current hierarchy tree or any combination of project design files and ancillary files.

Go to MAX+PLUS II Help for complete information on all MAX+PLUS II Hierarchy Display functions and features.

Project Processing

MAX+PLUS II processes projects for Altera Classic, MAX 5000, MAX 7000, MAX 9000, FLEX 6000, FLEX 8000, and FLEX 10K devices. MAX+PLUS II compiles projects automatically, but you can also make detailed processing specifications. Figure 2-18 shows how MAX+PLUS II compiles projects.

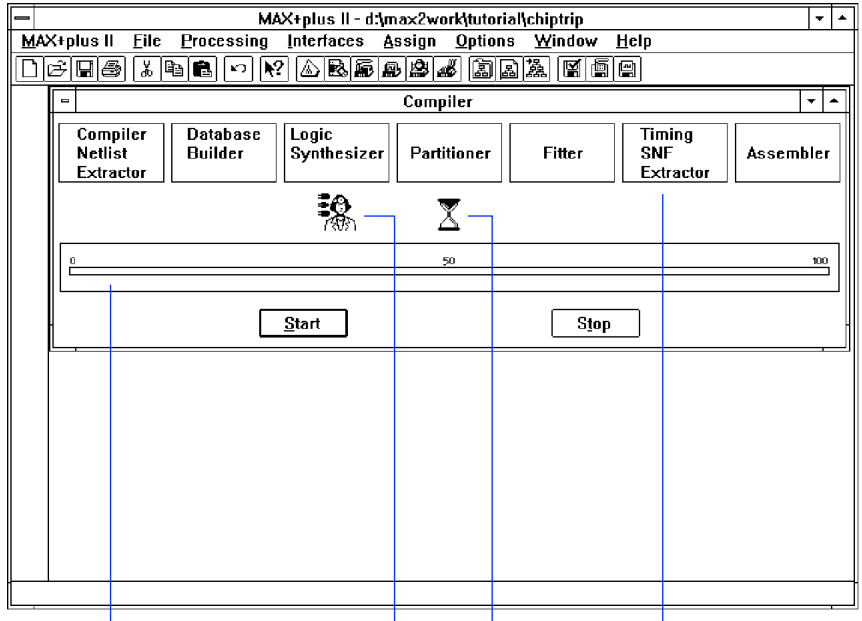
Figure 2-18. Project Processing



MAX+PLUS II Compiler

The MAX+PLUS II Compiler consists of a series of modules and a utility that check a project for errors, synthesize the logic, fit the project into one or more Altera devices, and generate output files for simulation, timing analysis, and device programming. The Compiler links the MAX+PLUS II design entry applications—the Graphic, Text, Waveform, Symbol, and Floorplan Editors—with the post-processing Timing Analyzer, Simulator, and Programmer applications. Figure 2-19 shows the MAX+PLUS II Compiler window.

Figure 2-19. MAX+PLUS II Compiler



Compiler Input Files

The MAX+PLUS II Compiler uses the following input files:

- Graphic Design Files (**.gdf**) created with the MAX+PLUS II Graphic Editor.
- Text Design Files (**.tdf**) created in the Altera Hardware Description Language (AHDL).
- VHDL Design Files (**.vhd**) created in VHDL 1987 or 1993 syntax.
- Verilog Design Files (**.v**) created in the Verilog HDL.
- EDIF version 2 0 0 or 3 0 0 Input Files (**.edf**) generated with any standard EDIF netlist writer.
- Waveform Design Files (**.wdf**) created with the MAX+PLUS II Waveform Editor.
- OrCAD Schematic Files (**.sch**) created with the OrCAD Draft schematic editor or with the MAX+PLUS II Graphic Editor.
- Xilinx Netlist Format Files (**.xnf**) created with Xilinx software.
- Altera Design Files (**.adf**) created with Altera's A+PLUS software. ADFs use a netlist format and Boolean equations to describe a design. The MAX+PLUS II Compiler automatically translates an ADF into a Compiler Netlist File (**.cnf**) during project compilation.
- State Machine Files (**.smf**) that contain a state machine design created for use with Altera's A+PLUS or SAM+PLUS software. The MAX+PLUS II Compiler automatically translates an SMF into an Altera Design File (**.adf**) and a Compiler Netlist File (**.cnf**) during compilation.
- Hexadecimal (Intel-format) Files (**.hex**) and/or Memory Initialization Files (**.mif**) containing the initial values for a memory block.
- An EDIF Command File (**.edc**) used to customize the format of EDIF Output Files (**.edo**) created by the MAX+PLUS II Compiler.

- An Assignment & Configuration File (**.acf**) that stores the project's probe, resource, and device assignments, as well as configuration settings for the Compiler, Simulator, and Timing Analyzer.

  Assignment and configuration information from pre-version 5.0 releases of MAX+PLUS II—which was stored in TDFs, Probe & Resource Assignment Files (**.prb**), and *<project name>***.ini** files—can be converted automatically to the ACF format.

- Symbol Files (**.sym**) created with the MAX+PLUS II Symbol Editor.

- Include Files (**.inc**) that are imported into an AHDL Text Design File (**.tdf**) by an AHDL Include Statement. The Include File replaces the Include Statement that calls it. Include Files can contain Function Prototype, Define, Parameters, or Constant Statements. The Compiler also uses AHDL Function Prototypes in Include Files to process logic functions in Verilog Design Files (**.v**).

- Library Mapping Files (**.lmf**) used to map cells in EDIF Input Files and OrCAD Schematic Files to corresponding MAX+PLUS II logic functions.

Compilation Process

The Compiler first extracts information that defines the hierarchical connections between a project's design files and checks the project for basic design entry errors. It creates an organizational map of the project and then combines all design files into a fully flattened database that can be processed efficiently.

The Compiler applies a variety of techniques to increase the efficiency of your project and minimize device resource usage. If your project is too large to fit into a single device, the Compiler can automatically partition it into multiple devices from the same device family, while minimizing the number of connections between devices. A Report File (**.rpt**) then shows how a project will be implemented in one or more devices.

The Compiler also creates programming files that the MAX+PLUS II Programmer or another industry-standard programmer uses to program one or more Altera devices.

While the Compiler can compile a project with minimal assistance, it also allows you to customize design processing to your exact specifications. For example, you can specify a default project logic synthesis style and other project-wide logic synthesis settings that tailor logic synthesis to your needs, enter project-wide timing requirements, specify precisely how to divide a large project into multiple devices, and set various device options on a project-wide basis. You can also choose how many pins and logic cells must remain unused during the current compilation to reserve additional logic capacity for future use.

Running the Compilation

You can start project compilation from any MAX+PLUS II application or from the Compiler. The Compiler automatically processes all input files for the current project, and you can monitor the compilation process in the Compiler window:

- The hourglass empties and flips, indicating that the Compiler is active.
- The module boxes are highlighted in turn as the Compiler completes each stage of processing.
- Icons representing output files appear below the boxes representing the Compiler modules that generate them. You can double-click Button 1 on an icon to open the corresponding file.
- The percent completion shown in the progress bar moves toward 100%.
- During partitioning and fitting, the Compiler's **Stop** button turns into a **Stop/Show Status** button, which you can choose to open a dialog box that shows the project's current partitioning and fitting status.
- If any errors or potential problems are detected during compilation, the Message Processor window opens automatically; lists information, error, and warning messages; and provides immediate help on how to correct an error; and allows you to locate message sources in the project's design files or its assignments floorplan.

The Compiler can run in the background. You can minimize it while it is processing a project, and continue working on other files. A progress bar under the minimized Compiler icon allows you to keep an eye on the compilation progress while you focus your attention on a different task.

Compiler Modules & Output Files

The MAX+PLUS II Compiler processes a project with the following modules and utilities:

- Compiler Netlist Extractor (including built-in EDIF, VHDL, Verilog, and XNF Netlist Readers)
- Database Builder
- Logic Synthesizer
- Partitioner
- Fitter
- Functional SNF Extractor
- Timing SNF Extractor
- Linked SNF Extractor
- EDIF Netlist Writer
- Verilog Netlist Writer
- VHDL Netlist Writer
- Assembler
- Design Doctor Utility

Compiler Netlist Extractor (Including Built-In EDIF Netlist Reader, VHDL Netlist Reader, Verilog Netlist Reader & XNF Netlist Reader)

The Compiler Netlist Extractor converts each design file in the project into one or more binary Compiler Netlist Files (**.cnf**). Because the Compiler Netlist Extractor resolves the values of any parameters used in parameterized functions, the contents of a CNF can change in a subsequent compilation if the parameter values change. The Compiler Netlist Extractor also creates a Hierarchy Interconnect File (**.hif**) that documents the hierarchical connections between the project files, and provides the information necessary to show the project's hierarchy tree in the Hierarchy Display. In addition, the Compiler Netlist Extractor generates the Node Database File (**.ndb**) that contains project node names for the resource assignment database.

The built-in EDIF, VHDL, Verilog HDL, and XNF netlist readers automatically translate the design information in EDIF Input Files (**.edf**), VHDL Design Files (**.vhd**), Verilog Design Files (**.v**), and Xilinx Netlist Format Files (**.xnf**), respectively, into a MAX+PLUS II-compatible format. The EDIF Netlist Reader processes EDIF Input Files with the help of Library Mapping Files (**.lmf**) that map logic functions provided with other industry-standard EDA tools to MAX+PLUS II functions. The XNF Netlist Reader optionally generates a Text Design Export File (**.tdx**) that contains the AHDL equivalent of a Xilinx Netlist Format File (**.xnf**) so that you can easily edit your project in AHDL.

Database Builder

The Database Builder uses the HIF to link the CNFs that describe the project. Based on HIF data, the Database Builder copies each CNF into a single, fully flattened project database. The database thus preserves the electrical connectivity of the project.

As it creates the database, the Database Builder examines the logical completeness and consistency of the project, and checks for boundary connectivity and syntactical errors (e.g., a node without a source or destination). Most errors are detected and can be easily corrected at this stage of processing. Each Compiler module subsequently processes and updates this database.

The first time the Compiler processes a project, all design files of that project are compiled. You can use the Compiler's "smart recompile" feature to create an expanded project database that helps to accelerate subsequent compilations. This database allows you to change physical device resource assignments, such as pin and logic cell assignments, and recompile the project without rebuilding the database and resynthesizing the project logic. With the "total recompile" feature, you can choose between recompiling only those files that have been edited since the last compile or fully recompiling the project.

Logic Synthesizer

The Logic Synthesizer module applies a number of algorithms that reduce resource usage and remove redundant logic to ensure that the logic cell structure is used as efficiently as possible for the architecture of the target device family. This Compiler module also applies logic synthesis techniques to help implement user-specified timing and other implementation requirements. In addition, the Logic Synthesizer searches the logic for unconnected nodes. If it finds an unconnected node, it removes the primitives associated with that node.

A large number of logic options, as well as three "ready-made" synthesis styles, are available to help you guide the outcome of logic synthesis.

You can enter timing assignments and logic option assignments, and define logic synthesis styles, in any MAX+PLUS II application. You can specify global default logic synthesis and timing options for an entire project, and entire additional logic option and timing assignments on individual logic functions.

Go to "Global Project Timing Requirements" and "Global Project Logic Synthesis" on page 100 for more information.

Partitioner

If a project does not fit into a single device, the Partitioner divides the database updated by the Logic Synthesizer into multiple devices from the same device family, attempting to split the project into the smallest possible number of devices. A project is partitioned along logic cell boundaries, and the number of pins used for inter-device communication is minimized.

Partitioning can be totally automatic, partially user-controlled, or fully user-controlled. Device assignments and automatic device selection settings allow you to exercise the level of control that is appropriate for your project.

While the Partitioner and Fitter modules are running, you can pause the compilation. The Compiler then displays information on the current status of the partitioning and fitting process, including a comparison of required and available resources, so that you can decide whether to continue the compilation.

Fitter

Using the database updated by the Partitioner, the Fitter matches the requirements of the project with the known resources of one or more devices. It assigns each logic function to the best logic cell location and selects appropriate interconnection paths and pin assignments. The Fitter attempts to match your resource assignments—i.e., the pin, logic cell, I/O cell, embedded cell, chip, clique, device, local routing, timing, and connected pin assignments in the project's Assignment & Configuration File (**.acf**)—with the available resources. The Fitter provides options to allow you to customize its fitting techniques to help achieve a fit, for example, by automatically inserting logic cells or limiting fan-in. If it cannot find a fit, the Fitter issues a message and gives you the option of ignoring some or all of your assignments or terminating compilation.

Regardless of whether a fit is achieved, the Fitter generates a Report File (**.rpt**) that documents fitting information on project partitioning, input and output pin names, project timing, and unused resources for each device in the project. You can optionally include Report File sections that show user assignments, file hierarchy, logic cell interconnections, and equations.

The Compiler also automatically generates a Fit File (**.fit**) that documents resource and device assignments for the entire project, as well as routing information. Regardless of whether a successful fit was achieved, you can view the fitting, partitioning, and routing information from the Fit File with the Floorplan Editor. You can also back-annotate Fit File assignments to the project's ACF for editing.

You can optionally direct the Fitter to generate AHDL Text Design Output Files (**.tdo**) for the fully optimized, fitted project. Since one file is generated for each device in a multi-device project, you can split your project into multiple single-device projects if you wish to "lock down" the logic in some of the devices. You can then edit the logic for a single device, save the TDO File for that device as a Text Design File (**.tdf**), and recompile the logic for that device while maintaining the logic synthesis results of a previous compilation.

Functional SNF Extractor

The optional Functional SNF Extractor creates the functional Simulator Netlist File (**.snf**) required for functional simulation. The Compiler generates this file before it synthesizes the project; therefore, it contains all nodes present in the original design files. The functional SNF does not contain timing information, but is generated quickly. The file is created only if a project compiles without errors.

Timing SNF Extractor

The optional Timing SNF Extractor creates the timing Simulator Netlist File (**.snf**), which contains the timing data for the fully optimized project. This file is used for timing simulation and timing analysis. The Compiler's EDIF Netlist Writer, Verilog Netlist Writer, and VHDL Netlist Writer modules also use timing SNFs to generate EDIF, Verilog HDL, and VHDL output files, as well as optional Standard Delay Format (SDF) Output Files (**.sdo**). The timing SNF is created only if a project compiles without errors.

You can optionally instruct the Compiler to generate an optimized SNF containing dynamic models that represent types of combinatorial logic. Optimizing the SNF increases the compilation time, but can save you time during simulation and timing analysis.

Linked SNF Extractor

The optional Linked SNF Extractor creates a linked Simulator Netlist File (**.snf**), which contains the functional and/or timing data for multi-project, board-level-type simulation. The linked SNF combines information from the timing SNFs and/or functional SNFs for multiple separate projects. Projects that are linked can use devices from different device families. If a linked SNF contains timing information only, you can also use it to run a timing analysis. The file is created only if a project compiles without errors.

Go to "Project Verification" on page 141 and to MAX+PLUS II Help for more information on the MAX+PLUS II Simulator.

EDIF Netlist Writer

The MAX+PLUS II Compiler can interface with most industry-standard CAE tools that can read a netlist file in the EDIF 2 0 0 or 3 0 0 standard format. The optional EDIF Netlist Writer produces one or more EDIF Output Files (**.edo**) containing post-synthesis functional and optional timing information. Timing information can also be written to separate Standard Delay Format (SDF) Output Files (**.sdo**). These files can be used with an industry-standard simulator. EDIF and SDF output files are created only if a project compiles without errors.

Verilog Netlist Writer

The optional Verilog Netlist Writer produces one or more Verilog Output Files (**.vo**) that contain the project's post-synthesis functional and optional timing information. Timing information can also be written to separate SDF Output Files. These files can be used with an industry-standard Verilog HDL simulator. Verilog HDL and SDF output files are created only if a project compiles without errors.

VHDL Netlist Writer

The optional VHDL Netlist Writer produces one or more VHDL Output Files (**.vho**), in VHDL 1987 or 1993 syntax, which contain the project's post-synthesis functional and optional timing information. Timing information can also be written to separate SDF Output Files. These files can be used with an industry-standard VHDL simulator. VHDL and SDF output files are created only if a project compiles without errors.

Assembler

The Assembler converts the Fitter's logic cell, pin, and device assignments into a programming image for the device(s) in the form of one or more binary Programmer Object Files (**.pof**) or SRAM Object Files (**.sof**); for some devices, the Compiler also generates JEDEC Files (**.jed**), Tabular Text Files (**.ttf**), and Hexadecimal (Intel-format) Files (**.hex**). The POFs, SOFs, or JEDEC Files are then processed by the MAX+PLUS II Programmer and Altera programming hardware, or another industry-standard programmer, to produce working devices. The Hex Files and TTFs can be used to configure FLEX 6000, FLEX 8000, and FLEX 10K devices by other means. The Assembler creates programming files only if the project compiles without errors.

After compilation has been completed, the MAX+PLUS II Compiler and Programmer allow you to generate additional device programming files for use in other programming environments. For example, you can create Serial Bitstream Files (**.sbf**) and Raw Binary Files (**.rbf**) for configuring FLEX 6000, FLEX 8000, and FLEX 10K devices. You can also create Serial Vector Format Files (**.svf**) and Jam Files (**.jam**) for programming devices in automated test equipment (ATE)-type and embedded processor-type programming environments, respectively.

Design Doctor Utility

The optional Design Doctor utility checks each design file for logic that may cause system-level reliability problems that are usually discovered only after a design has entered production. You can choose one of three predefined sets of design rules with different levels of design-rule checking, or create a custom set of design rules.

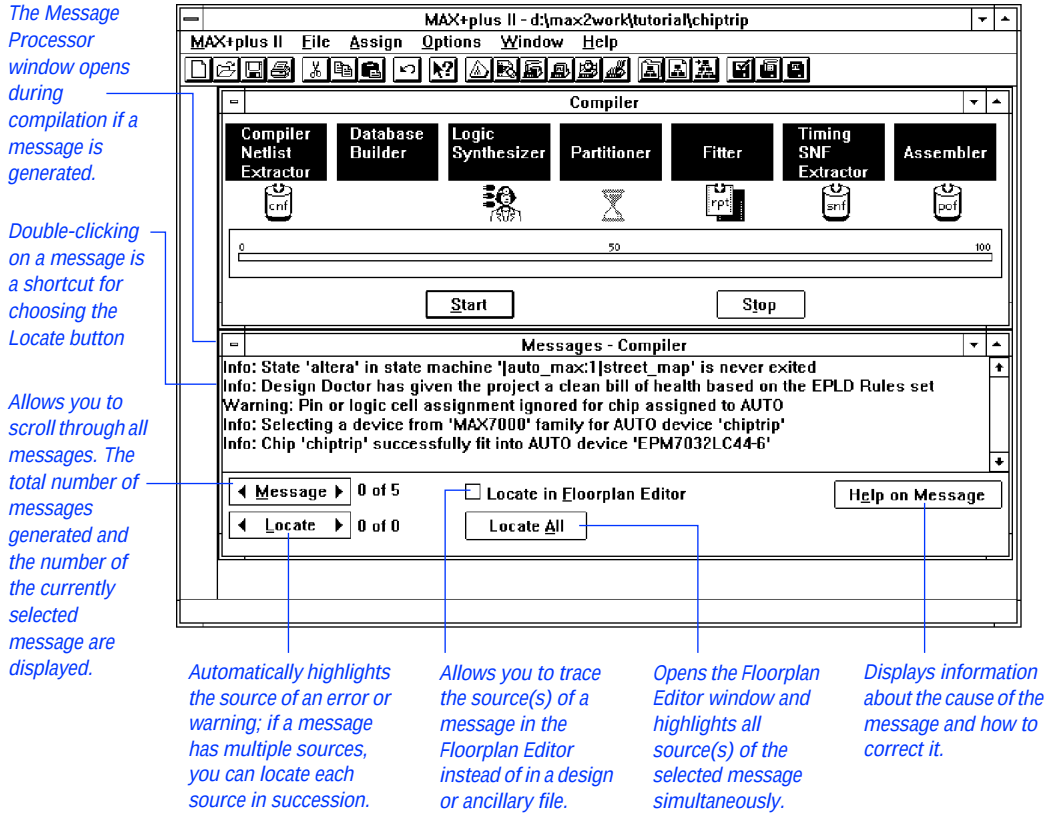
Design rules are based on reliability guidelines that cover logic containing features such as asynchronous inputs, ripple Clocks, multi-level logic on Clocks, Preset and Clear configurations, and race conditions. MAX+PLUS II Help explains rule violations to help you determine which edits are needed in the design files.

Go to MAX+PLUS II Help for complete information on all Compiler functions and features.

Error Detection & Location

The MAX+PLUS II Message Processor communicates with all MAX+PLUS II applications, recording error, information, and warning messages as you process, verify, or program a project. If you double-click Button 1 on a message, the Message Processor automatically opens the design file, ancillary file, or the project floorplan that contains the source of the message, and highlights the location of the error. See Figure 2-20.

Figure 2-20. MAX+PLUS II Message Processor



The Message Processor offers the following features:

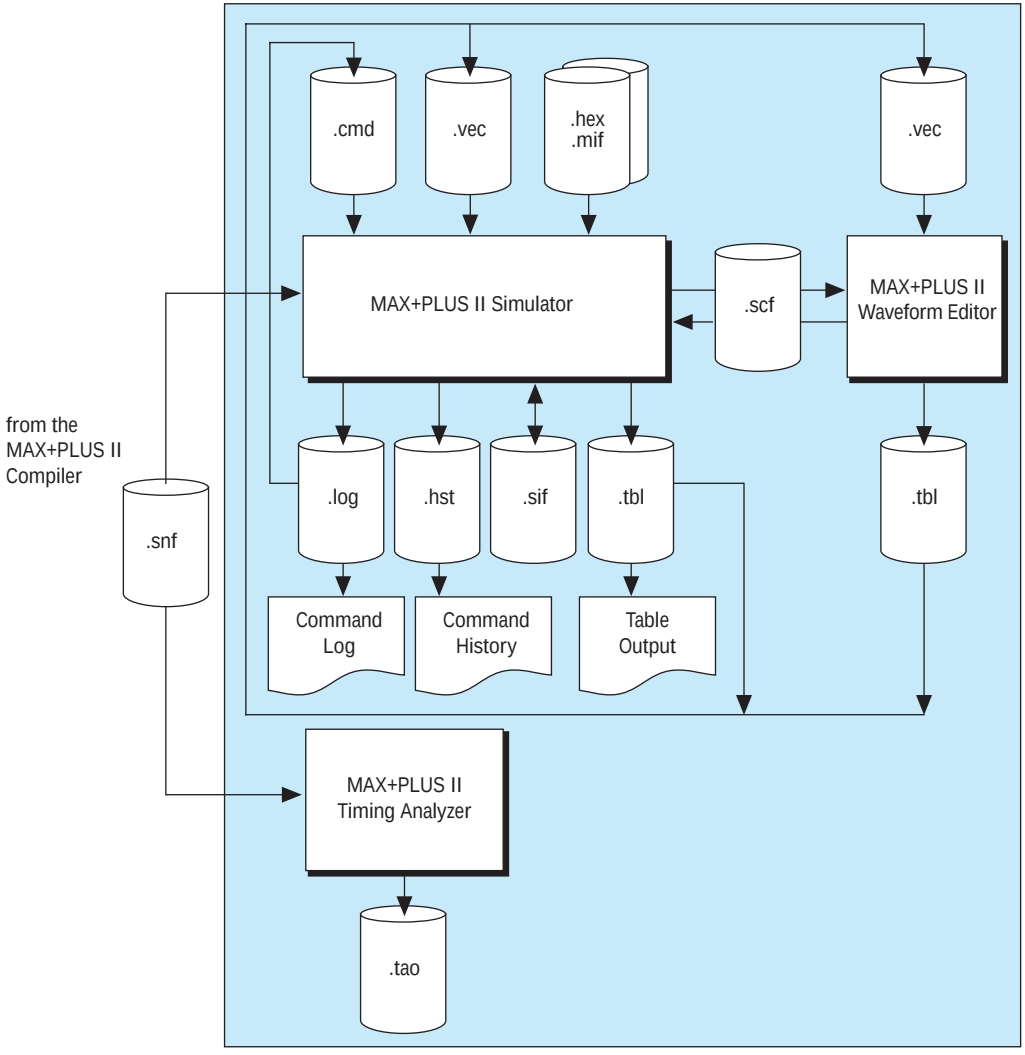
- If a message is caused by a problem in multiple locations, the Message Processor allows you to find each location.
- You can locate the source of a message in the project design files and ancillary files (e.g., in a Simulator Channel File), or locate to the floorplan for the project.
- When you list signal paths with the **List Paths** button in the MAX+PLUS II Timing Analyzer, the Message Processor displays messages that show all propagation delays between a pair of nodes.
- If you choose to locate messages in the project floorplan, you can locate each source of a message in succession, or locate all sources simultaneously. Locating all sources simultaneously provides a convenient method for viewing critical timing paths.
- You can print the current messages or save the messages in an ASCII Message Text File (**.mtf**).
- When a message is highlighted in the Message Processor window and you choose the **Help on Message** button, you get detailed information on the likely cause of the message and the suggested corrective action.

Go to MAX+PLUS II Help for complete information on all Message Processor functions and features.

Project Verification

MAX+PLUS II provides three applications—the MAX+PLUS II Simulator, Timing Analyzer, and Waveform Editor—to help you test the logic of a compiled project. See Figure 2-21.

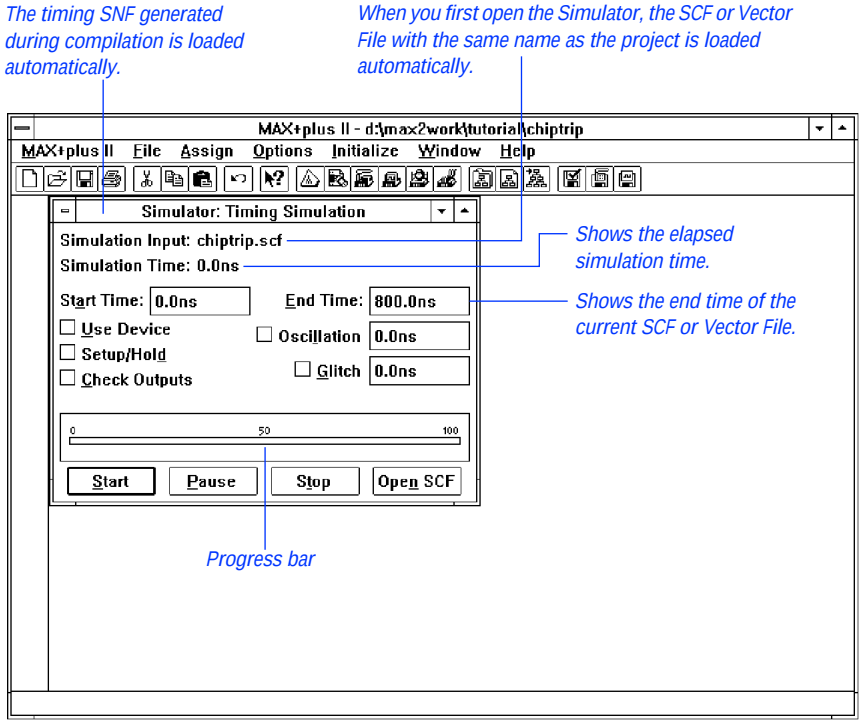
Figure 2-21. MAX+PLUS II Project Verification



MAX+PLUS II Simulator

The MAX+PLUS II Simulator tests the logical operation and internal timing of a project, allowing you to model a circuit design before it is programmed into a device. You can run the Simulator either in interactive mode or in batch mode. Figure 2-22 shows the Simulator window.

Figure 2-22. MAX+PLUS II Simulator



To simulate a project, you must first compile it and instruct the Compiler to generate a Simulator Netlist File (**.snf**) for functional, timing, or linked multi-project simulation. The functional, timing, or linked SNF for the current project is then loaded automatically when you open the Simulator.

The Simulator uses a graphical waveform Simulator Channel File (**.scf**) or an ASCII Vector File (**.vec**) as the source of input vectors. For projects that contain memory, you can specify the initial memory contents with Hexadecimal (Intel-format) Files (**.hex**) or with Memory Initialization Files **(.mif**). The Waveform Editor can automatically create a default SCF, which you can edit to provide the desired input vectors; if you create a Vector File instead, the Simulator automatically generates an SCF from it.

Go to *Vector File Format* in MAX+PLUS II Help for a description of the Vector File format. You should also try out "Session 10: Simulate the Project" on page 255 in the *MAX+PLUS II Tutorial*.

The Simulator allows you to check the outputs of simulation against any outputs in the SCF, such as user-defined expected outputs or outputs from a previous simulation. With appropriate programming hardware, you can also perform functional testing to test the actual outputs of a programmed device against the simulation outputs.

Using options in the Simulator, you can monitor your project for glitches, oscillations, and setup and hold time violations. Once simulation is completed, you can open the Waveform Editor to view the updated SCF or save the outputs to a Table File (**.tbl**) and view the results in the Text Editor.

Functional Simulation

When the MAX+PLUS II Compiler creates a functional SNF, it generates the SNF before it synthesizes the project. Consequently, in a functional simulation, all nodes in the project can be simulated.

During functional simulation, the Simulator ignores all propagation delays. Because there are no delays in the functional SNF, output logic levels change at the same time as the input vectors.

Timing Simulation

When the MAX+PLUS II Compiler creates a timing SNF, it generates the SNF after the project has been fully synthesized and optimized. Therefore, a timing SNF contains only those nodes that have not been eliminated during logic synthesis.

The Simulator uses the information from the timing SNF, which contains hardware information from Device Model Files (**.dmf**) provided with MAX+PLUS II, to simulate the project.

If a project has been partitioned into two or more devices, the Compiler creates an SNF for the project as a whole and for each device. However, timing simulation is performed for the entire project only.

You can accelerate timing simulation by instructing the Compiler to generate an optimized SNF containing dynamic models that represent various types of combinatorial logic. The Compiler's processing time increases when it generates the optimized SNF; however, the resulting SNF can reduce simulation time, because the Simulator can refer to a representative dynamic model instead of interpreting all logic in a combinatorial network.

Linked Multi-Project Simulation

When the MAX+PLUS II Compiler creates a linked SNF, it combines the functional and/or timing SNFs for multiple individual projects. The separate "sub-projects" in the linked SNF can be targeted for different Altera device families. In addition, because functional SNFs are not fully compiled, you can incorporate sub-projects that represent logic that is not implemented in an Altera device.

You can use the linked SNF to perform a board-level-type simulation. In addition, if the linked SNF contains timing information only, you can use it to run a timing analysis in the MAX+PLUS II Timing Analyzer.

Simulator Highlights

Together with other MAX+PLUS II applications, the Simulator lets you perform the following tasks:

- Specify expected output logic levels that can be compared to simulation outputs.
- Simulate individual or grouped nodes. You can combine bits of a state machine in the project, simulate them as a group, and refer to them with a state name.

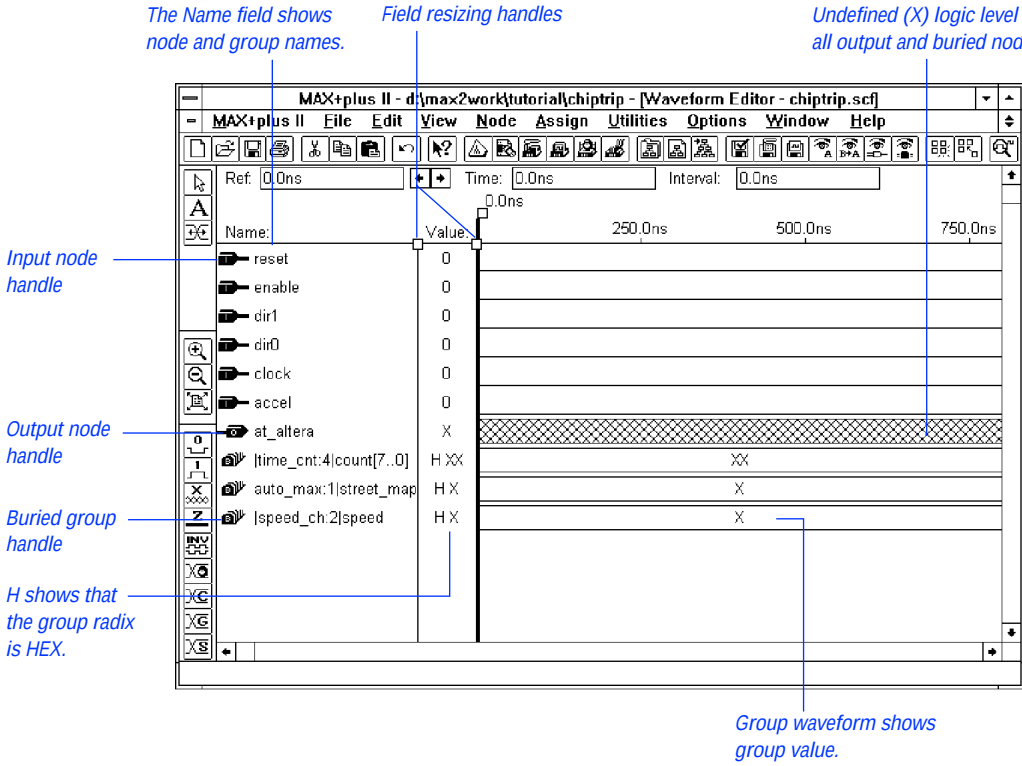
- Define a time interval that constitutes an oscillation or glitch, and analyze the project for one or both conditions.
- Monitor the project for register setup and hold time violations.
- Record actual device outputs instead of simulation outputs.
- Perform functional testing. You can check whether simulation outputs are functionally equivalent to actual device outputs.
- Create breakpoint conditions that cause the Simulator to pause when these conditions are met during simulation.
- List the name and logic level of any combination of nodes and groups and initialize node and group logic levels before simulation.
- Initialize the contents of memory blocks (RAM or ROM) before simulation.
- Save initialized node and group values, including initialized memory values, in a Simulator Initialization File (**.sif**), or reload the initialized values stored in the file.
- Log Simulator commands to an ASCII Log File (**.log**) with the same format as the Command File (**.cmd**) used for batch-mode simulation, and use the Log File to repeat an earlier simulation. You can also record Simulator commands and their output in an ASCII History File (**.hst**).

Go to MAX+PLUS II Help for complete information on all MAX+PLUS II Simulator functions and features.

MAX+PLUS II Waveform Editor

The MAX+PLUS II Waveform Editor is used as a design entry tool and as a tool for entering input vectors and viewing simulation results. You can create Waveform Design Files (**.wdf**) that contain design logic for the project and Simulator Channel Files (**.scf**) that contain input vectors for simulation. See Figure 2-23.

Figure 2-23. MAX+PLUS II Waveform Editor



To simulate a project, you must provide input vectors. With an SCF, you can describe simulation input vectors as waveforms, which are a graphical alternative to the ASCII Vector File (**.vec**) input for the Simulator.

You create an SCF containing the input vector waveforms that will drive simulation, and the buried node and output node names to be simulated. Buried and output nodes have undefined logic levels, or can be edited to include expected logic values. The Waveform Editor can use the Simulator Netlist File (**.snf**) to create a default SCF that contains some or all nodes and groups in the compiled project. You can edit this SCF to meet your specifications or you can create an SCF "from scratch." In addition, you can import a Vector File to automatically create its graphical waveform equivalent.

With the Waveform Editor, you can view and interpret the outputs of simulation in an SCF. The Simulator generates an SCF automatically if a Vector File is the source of input vectors; otherwise, the SCF that was the source of vectors is simply updated during simulation. Buried logic and output node waveforms are overwritten with logic levels based on simulation inputs.

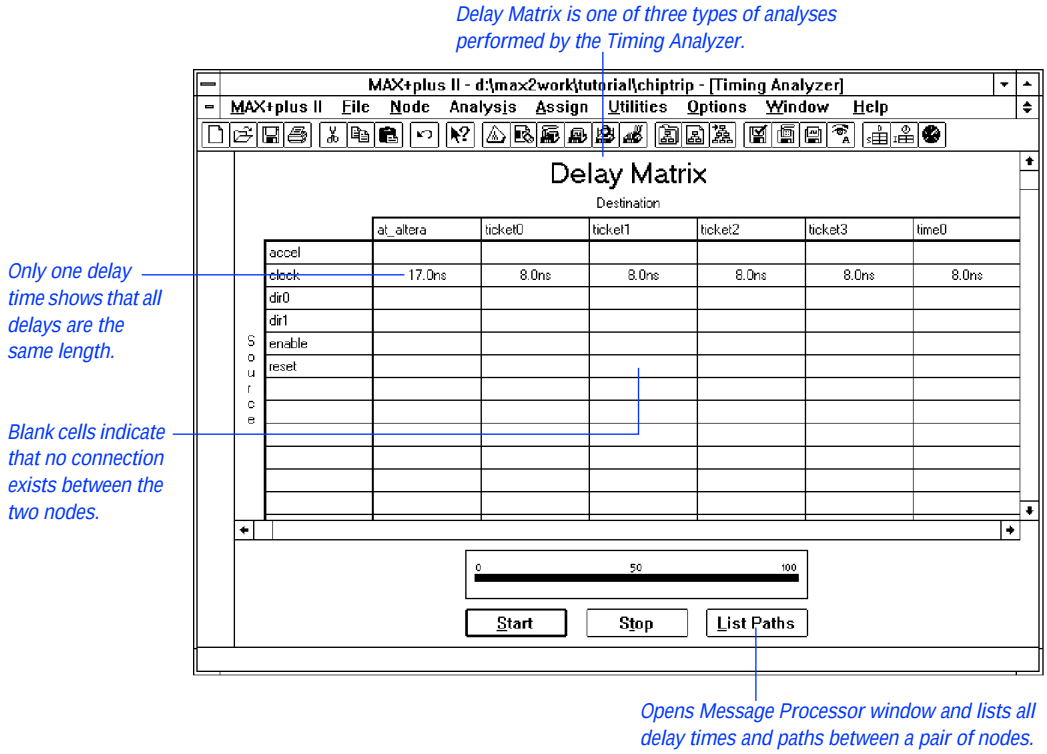
For more details, go to "MAX+PLUS II Waveform Editor" on page 146.

Go to MAX+PLUS II Help for complete information on the MAX+PLUS II Waveform Editor.

MAX+PLUS II Timing Analyzer

With the MAX+PLUS II Timing Analyzer, you can analyze the timing performance of a project after it has been optimized by the Compiler. You can trace all signal paths in the project, determining critical speed paths and paths that limit the project's performance. See Figure 2-24.

Figure 2-24. MAX+PLUS II Timing Analyzer



The Timing Analyzer uses the network and timing information from a timing Simulator Netlist File (**.snf**) generated by the Compiler. The Timing Analyzer can also use a linked SNF that links the timing SNFs of other projects.

The Timing Analyzer generates three types of analyses:

- The *Delay Matrix* shows the shortest and longest propagation delay paths between multiple source and destination nodes in a project.
- The *Setup/Hold Matrix* shows the minimum required setup and hold times from input pins to the data, Clock, Clock Enable, Latch Enable, address, and Write Enable inputs to flipflops, latches, and asynchronous RAM.
- The *Registered Performance Display* shows the results of a registered performance analysis, including a user-defined number performance-limiting delays, minimum Clock period, and maximum circuit frequency.

With the Timing Analyzer, you can tag multiple source and destination nodes so that they are included in an analysis. You can tag these nodes directly in design files in the Graphic, Text, and Waveform Editors, and in the project floorplan with the Floorplan Editor. You can also use the default timing tagging available for each type of analysis. Moreover, you can specify maximum and minimum delays, and completely cut off a signal path from the timing analysis so that only the signal that leads to the node is included in the analysis. You can also exclude I/O pin feedback, Clear, and Preset signal delay paths from the analysis, and restrict the number of paths to list per Clock in a Registered Performance analysis.

After the Timing Analyzer completes an analysis, you can select a source or destination node and list all delay paths associated with it. The Message Processor automatically opens and lists the paths for the selected node, so that you can locate a specific path in the original design file or in the project floorplan.

You can save the results of a timing analysis to a Timing Analyzer Output File (**.tao**).

Go to MAX+PLUS II Help for complete information on all MAX+PLUS II Timing Analyzer functions and features.

Device Programming

Altera provides all hardware and software necessary for programming, configuring, and verifying Altera devices. The hardware includes an add-on Logic Programmer card (for 486- or Pentium-based PCs) that drives the Altera Master Programming Unit (MPU). The MPU performs continuity checking to ensure adequate electrical contact between the programming adapter and the device. With the appropriate programming adapter, the MPU also supports functional testing, so that you can apply vectors created for simulation to a programmed device to verify its functionality.

Altera also supports in-system programmability (ISP) and in-circuit reconfiguration (ICR) with the FLEX Download Cable (for PCs), the ByteBlaster parallel download cable (for PCs), and the BitBlaster serial download cable (for PCs and UNIX workstations). The FLEX Download Cable can connect any Configuration EPROM programming adapter, which is installed on the MPU, to target FLEX devices in a prototype system. The ByteBlaster connects a parallel port (i.e., printer port), and the BitBlaster serial download cable connects a standard RS-232 port (called a "COM port" on a PC), to devices mounted on system boards. These cables allow you to program or configure one or more ICR- or ISP-compatible devices in a FLEX chain or a JTAG chain. See Table 2-4.

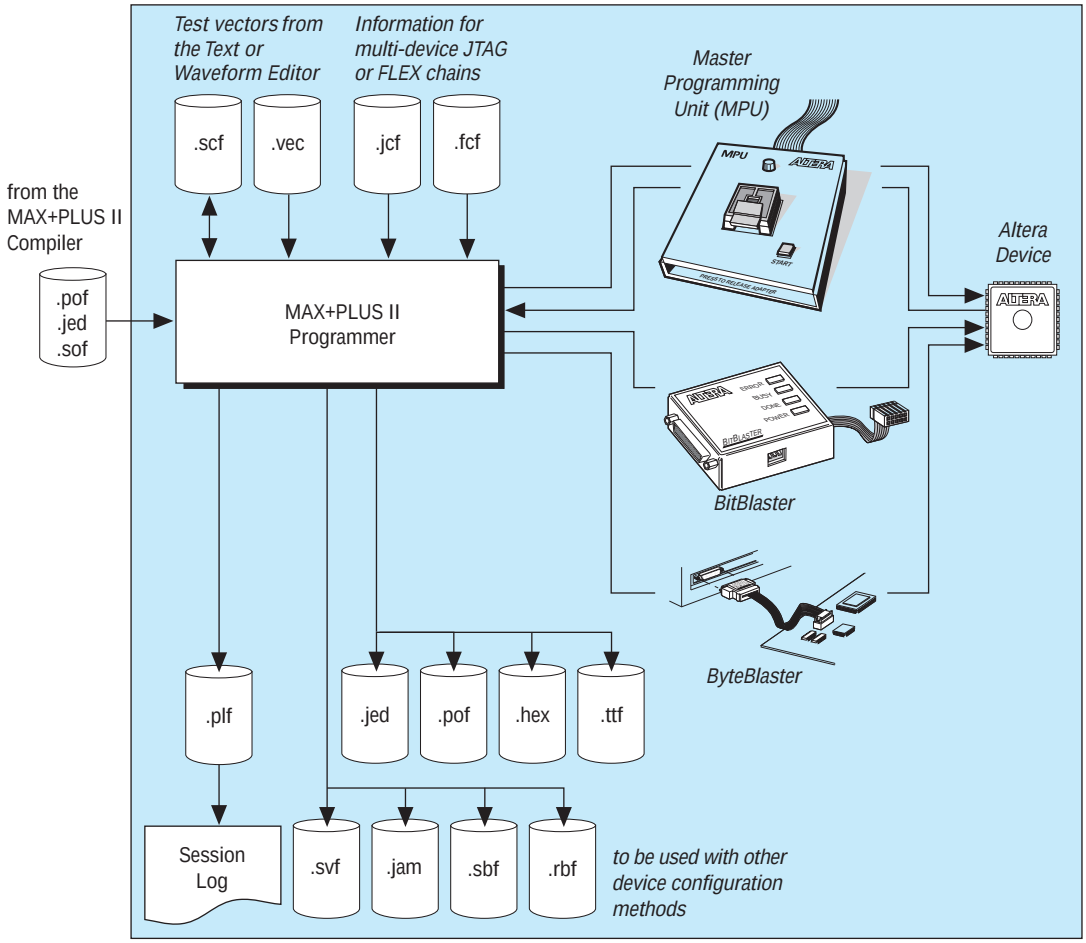
Table 2-4. Altera Programming Hardware

Programming Hardware Option	PCs	UNIX Workstations	Classic & MAX 5000 Devices	MAX 7000 Devices (excluding MAX 7000S)	MAX 7000S & MAX 9000 Devices	FLEX 6000, FLEX 8000 & FLEX 10K Devices	In-System Programming/ Configuration
Logic Programmer card, PL-MPU Master Programming Unit, and device-specific adapters	✓		✓	✓	✓		
FLEX Download Cable	✓					✓	✓
BitBlaster Download Cable	✓	✓			✓	✓	✓
ByteBlaster Download Cable	✓				✓	✓	✓

Go to "Installing the Programming Hardware" on page 53 for more information about Altera programming hardware.

Go to *Application Note 87 (Configuring FLEX 6000 Devices)*, *Application Note 33 (Configuring FLEX 8000 Devices)*, *Application Note 38 (Configuring Multiple FLEX 8000 Devices)*, and *Application Note 59 (Configuring FLEX 10K Devices)*, for instructions on how to configure SRAM-based FLEX 6000, FLEX 8000, and FLEX 10K devices.

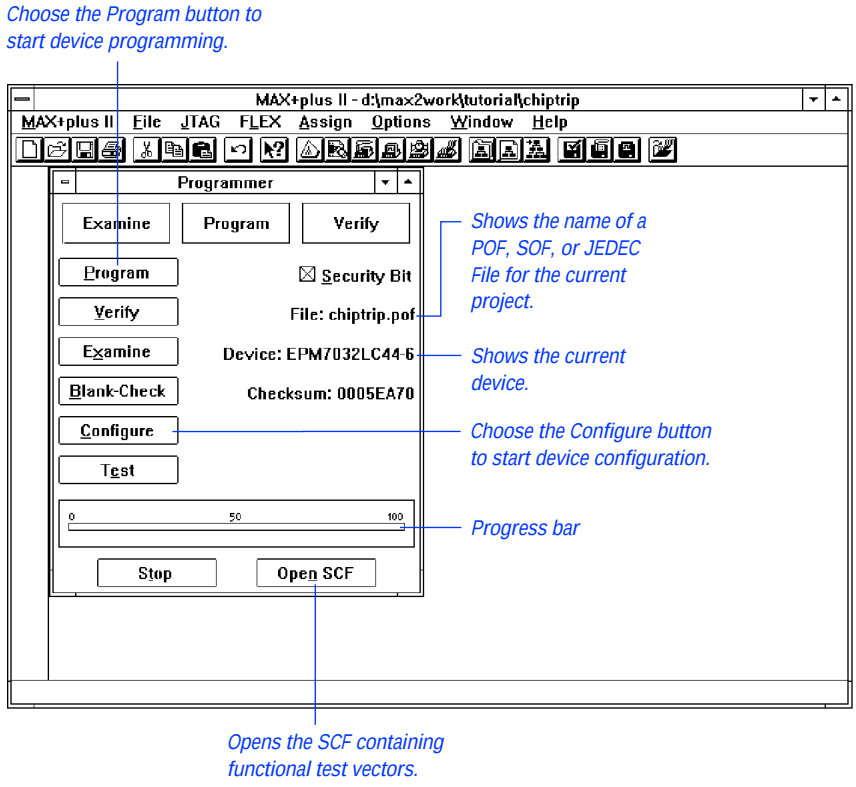
Figure 2-25. MAX+PLUS II Device Programming



MAX+PLUS II Programmer

The MAX+PLUS II Programmer, shown in Figure 2-26, uses programming files generated by the Compiler to program Altera devices. It allows you to program, configure, verify, examine, blank-check, and functionally test devices.

Figure 2-26. MAX+PLUS II Programmer



Altera programming hardware enables you to program Altera devices with the programming file(s) generated by the Compiler. When you open the Programmer window, the programming file for the current project is automatically loaded. The MAX+PLUS II Programmer accepts the following file formats:

- A Programmer Object File (**.pof**) to program Altera Classic, MAX 5000, MAX 7000, and MAX 9000 devices, as well as the Configuration EPROMs used to configure FLEX 6000, FLEX 8000, and FLEX 10K devices.
- An SRAM Object File (**.sof**) to configure Altera FLEX 6000, FLEX 8000, and FLEX 10K devices.
- A JEDEC File (**.jed**) to program Altera Classic devices and the EPM5016 and EPM5032 devices from the MAX 5000 family.
- A JTAG Chain File (**.jcf**) that describes the order in which POFs, SOFs, and JEDEC Files for multiple devices on a circuit board are to be programmed or configured in a chain of multiple devices that are connected by JTAG circuitry.
- A FLEX Chain File (**.fcf**) that describes the order in which SOFs for multiple FLEX devices on a circuit board are to be configured in a FLEX chain.

The Programmer and Compiler can also generate Hexadecimal (Intel-Format) Files (**.hex**), Tabular Text Files (**.ttf**), Serial Bitstream Files (**.sbf**), Raw Binary Files (**.rbf**), Serial Vector Format Files (**.svf**), and Jam Files (**.jam**) for configuring and programming ISP- and ICR-compatible devices in other programming environments.

You start device programming or configuration by choosing the **Program** or **Configure** button. If any errors or problems are detected during programming or configuration, the Message Processor window lists messages and provides immediate help on how to correct an error.

The MAX+PLUS II Programmer can perform the following tasks:

- Programs POF or JEDEC File data into a blank Classic, MAX 5000, MAX 7000, or MAX 9000 device to produce a working device. In multi-device JTAG chain mode, multiple devices are programmed with additional information from a JCF.
- Downloads configuration data from an SRAM Object File (**.sof**) or a JEDEC File (**.jed**) to configure FLEX 6000, FLEX 8000, or FLEX 10K devices. In multi-device JTAG chains, multiple devices are configured with additional information from a JCF. In multi-device FLEX chains, devices are configured with additional information from an FCF.

- Creates and reads JCFs and FCFs that detail the order of devices that will be programmed or configured in a multi-device JTAG or FLEX chain, respectively.
- Converts a POF into JEDEC File format or vice versa, and optionally saves functional testing vectors in the file, so that you can program and test a device with other industry-standard programming hardware and software.
- Verifies the programming file contents against the contents of a programmed device.
- Examines a programmed device and saves the programming data and test vector data in POF or JEDEC File format.
- Checks to ensure that an erasable device is blank or completely erased.
- Turns the Security Bit on or off before project data is programmed into a device. When the Security Bit is on, the device cannot be interrogated. EPROM-based devices also cannot be reprogrammed.
- Tests a programmed device with the input vectors in the current programming file, SCF, or Vector File, and reports whether the output logic levels in the file are functionally equivalent to actual device outputs.
- Optionally creates an output Programmer Log File (**.plf**) that records programming session commands and messages for future reference.

Go to MAX+PLUS II Help for complete information on all Programmer functions and features.

Section

3

MAX+PLUS II Tutorial

This tutorial demonstrates the basic features of MAX+PLUS II.

Introduction

MAX+PLUS II is easy—easy to learn, easy to use, and very easy to like. This tutorial introduces you to the basic features of the fully integrated MAX+PLUS II design environment, so you'll be able to create your own logic designs in record time. Once you start using MAX+PLUS II, the on-line help (always just a mouse-click away) can fill in all the details.

In this tutorial, you will create a design (called a "project" in MAX+PLUS II) named **chiptrip**, a simple driving simulator. After you enter and compile the **chiptrip** project, you will simulate it. In the simulation sessions, you will guide your "vehicle" through an imaginary street map. Your challenge will be to drive from your company to Altera using the most direct route without getting tickets from the police. Once you finish the simulation task, your final step will be to program your completed project into an Altera device.

The tutorial is divided into four sections: creating the actual logic circuit, compiling it, simulating it with multiple sets of inputs, and then programming an Altera device. To accommodate your level of expertise and to make sure that you experience some driving pleasure on the way (remember *Fahrvergnügen*?), all files for this project are provided in the **\max2work\chiptrip** directory. Thus, you can choose to go through every single step of the tutorial or take one or more shortcuts by copying the ready-made files to your working directory. Since the tutorial is divided into logical chunks, you can stop at any time and continue later. Have a good trip!

Project Description

The **chiptrip** tutorial takes you through all major steps of design entry, compilation, simulation, and programming for a hierarchical project.

Design Entry & Project Processing

You will create five design files using text, graphic, and waveform design entry. This tutorial describes a "bottom-up" hierarchical design entry method, in which you create the lower-level designs first and then combine them in a single top-level design file to create the **chiptrip** project. A project consists of all files associated with a particular design, including all subdesign files and ancillary files; the project name is always the same as the name of the top-level design file, without the filename extension. In the **chiptrip** project, the top-level Graphic Design File (**.gdf**), **chiptrip.gdf**, incorporates four lower-level design files—a GDF, two Text Design Files (**.tdf**), and a Waveform Design File (**.wdf**). Each lower-level file performs a specific function in the driving simulation game:

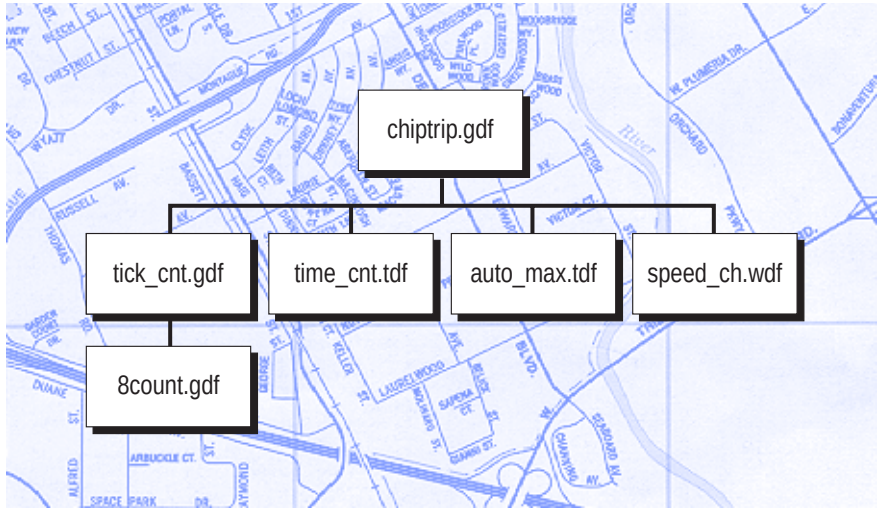
- The **tick_cnt.gdf** file, your "driving record," counts the number of police citations you collect as you drive. This counter adds up the number of tickets issued for "illegal" speeds in **auto_max.tdf** and **speed_ch.wdf**.
- The **time_cnt.tdf** file, the "clock" in your car, counts the number of clock pulses required for the vehicle to reach Altera.
- The **auto_max.tdf** file, your "automobile," contains a state machine that monitors the direction and acceleration inputs to the project and determines the next location (i.e., state) of the vehicle.

- The **speed_ch.wdf** file, your "speedometer," is a state machine that checks the acceleration of the vehicle. Illegal speeds result in a speeding ticket.

☞ If you have not purchased the waveform design entry feature for MAX+PLUS II, you can use a TDF version of the **speed_ch.wdf** file, called **speed_ch.tdf**. This file is available in the **\max2work\chiptrip** subdirectory.

Figure 3-1 shows a block diagram of the **chiptrip** project:

Figure 3-1. Block Diagram of chiptrip

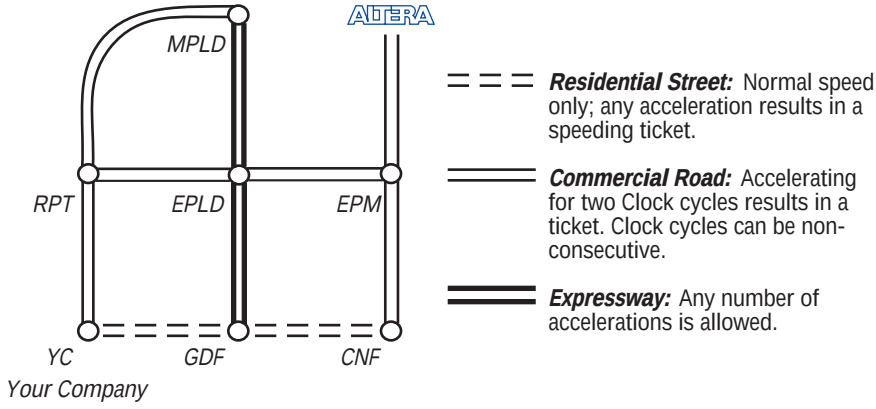


After you have created the design files, you must successfully compile your project to generate the files you need to simulate **chiptrip** and program a device.

Project Verification & Device Programming

The simulation portion of the tutorial is a driving game. The game tests your ability to plan and modify your simulation inputs to complete a specific task. Your goal is to navigate your vehicle through different intersections on a map (shown in Figure 3-2) to arrive at Altera as fast as you can and with as few speeding tickets as possible. Depending on how you edit your simulation inputs, you can maneuver your car along expressways, commercial roads, or residential streets.

Figure 3-2. Map to Altera



On expressways, you can go as fast as you like without worrying about any police officers stopping you. On commercial roads, you can accelerate once without getting a speeding ticket, but you will definitely get caught the second time. If you accelerate at all on residential streets, however, you will get a ticket right away. Just remember, in this design logic universe, police officers are everywhere, they always know when you are speeding, and you can't talk them out of giving you a ticket.

After you practice simulating your project with multiple sets of input vectors and analyzing its timing to your satisfaction, you can then program the **chiptrip** project into an Altera device.

Tutorial Overview

The **chiptrip** tutorial is designed to help you become an expert MAX+PLUS II user quickly and easily. The tutorial is modular, so you can complete the sessions at your own pace; work through one session at a time, or the whole tutorial in one sitting. You can also adapt the tutorial to your level of expertise. For example, if you feel comfortable with the various design entry methods, you can skip one or more of the sessions and move straight on to compiling and simulating your project. In addition, Sessions 5 and 9 introduce you to command shortcuts that can help you develop more efficient design entry skills.

Tutorial Files

All tutorial files are copied to your hard disk during MAX+PLUS II installation. The MAX+PLUS II working directory, which has the default name **\max2work**, contains the **chiptrip** and **tutorial** subdirectories. The **\max2work\chiptrip** subdirectory contains all design files, as well as user- and MAX+PLUS II-generated files for this tutorial. To prevent changes to the original files, you should create your project in the **\max2work\tutorial** subdirectory. If you do not wish to create an entire design file from scratch, you can simply copy the desired file from the **\max2work\chiptrip** subdirectory into the **\max2work\tutorial** subdirectory without running the risk of accidentally overwriting the original tutorial files installed on your hard disk. You can copy files with the appropriate copying command for your operating system, or open a file in MAX+PLUS II and choose **Save As** (File menu) to save a copy of the file in a different directory.

1. Be sure to refer to the **read.me** file in the **\max2work\tutorial** directory for information on changes to the **chiptrip** tutorial since this manual was printed.

2. On a UNIX workstation, the **max2work** directory is a subdirectory of the **/usr** directory.

Command Shortcuts

Many MAX+PLUS II commands have a variety of shortcuts. These shortcuts are often context-sensitive, that is, the options available depend on the position of the mouse pointer or on the item(s) selected on screen. Although you can use shortcuts at any stage of the tutorial process, Sessions 5 and 9 provide you with specific alternative steps to help speed up design entry.

You can experiment with these shortcuts and determine which one(s) you prefer. The shortcuts will help you develop an efficient and personalized method for working with the MAX+PLUS II software.

The "Shortcuts" section of MAX+PLUS II Help for each application lists all Button 1, keyboard, and toolbar/tool palette shortcuts.

Shortcut Method:	Description:
 Mouse Button 1	Button 1 shortcuts, which are executed by double-clicking, are context-sensitive. For example, in the Graphic Editor, you can open the **Enter Symbol** dialog box by simply double-clicking Button 1 in a blank space in the window. In contrast, double-clicking Button 1 on a macrofunction symbol opens the macrofunction that the symbol represents. Button 1 corresponds to the left button on a two- or three-button mouse.
 Mouse Button 2	Button 2 shortcuts, which are executed by clicking to display a pop-up menu, are also context-sensitive. These shortcuts allow you to execute a task by pointing to a selection, pressing Button 2, and choosing a command as you work. For instance, you can cut a selected object or a section of text out of a file by clicking Button 2 on the selected item and choosing **Cut** from the pop-up menu. Button 2 corresponds to the right button on a two-button mouse or the middle and right buttons on a three-button mouse.
 Keyboard	Keyboard shortcuts allow you to perform a task instantly. For example, typing Ctrl+P is a keyboard shortcut for the **Print** command. Keyboard shortcuts are listed in the pull-down menus and in MAX+PLUS II Help.
 Toolbar/Palette	Toolbar and tool palette shortcut buttons are available on the top and left sides of the window. For example, choosing the **Zoom In** button on the tool palette is a shortcut for the **Zoom In** command.

Getting Help

Throughout the tutorial, you can follow the footprints ( ) for useful references to MAX+PLUS II Help. On-line help provides the most up-to-date and complete information on all MAX+PLUS II features. Two of the easiest ways to get on-line help are by using the context-sensitive help feature and the search index.

Context-Sensitive Help

Context-sensitive help gives you instant help when you need it. You can access context-sensitive help in three ways:

Method:	Description:
Shift+F1 *or* the context-sensitive help toolbar button	Position the question-mark pointer over an item on the screen, a text file keyword, or a menu command, then click Button 1 on it to obtain help.
F1 key	When a menu command is highlighted, a dialog box is open, or a pop-up message box is displayed, press F1 to obtain help. You can also press F1 when any MAX+PLUS II application window is displayed to obtain general information about the context-sensitive help available for that application.
Help on Message button	In the Message Processor window, you can select a message with Button 1 and choose the **Help on Message** button to display help on that message.

Search Index

MAX+PLUS II Help includes an extensive on-line index to help you find information fast. To search for a Help topic:

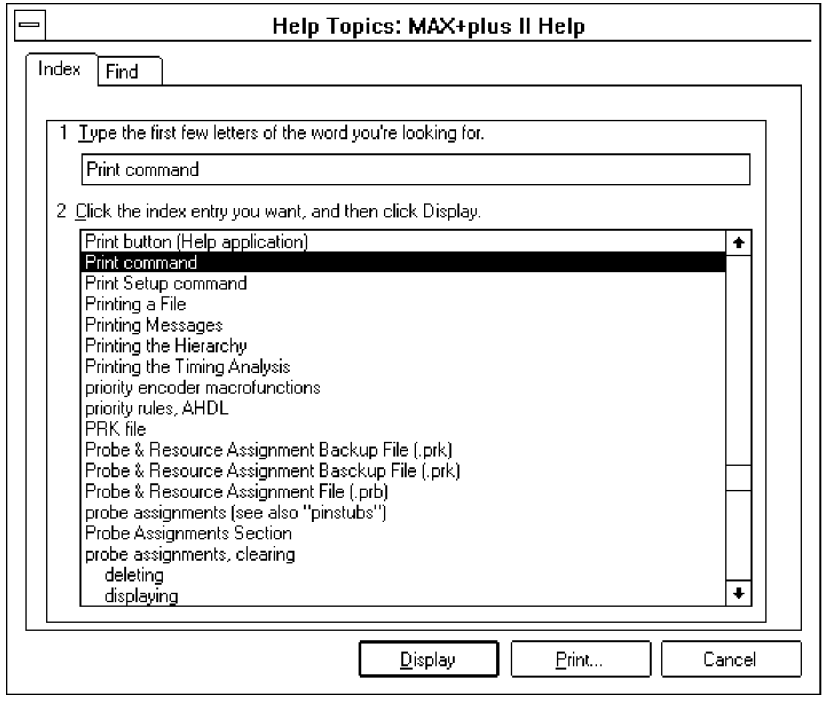
1. If you are in MAX+PLUS II, choose **Search for Help on** (Help menu).

   or:

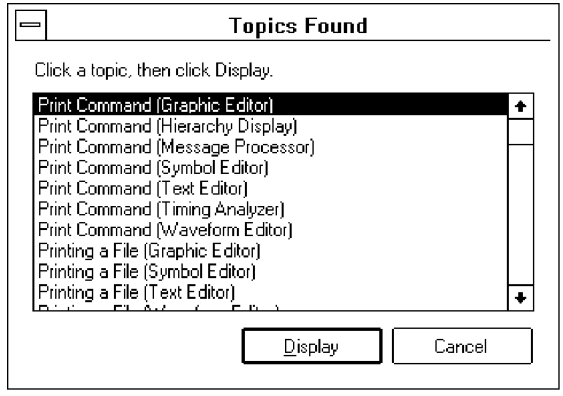
   If you are already in Help, choose the **Index** button at the top of the Help window (called **Search** in Windows NT 3.51).

   The **Help Topics** dialog box is displayed. (In Windows NT 3.51, the **Search** dialog box, which functions in a similar manner, appears instead of the **Help Topics** dialog box.)

2. Type a keyword or phrase. The keyword list scrolls to display the keywords that match the text you type, as shown in the following illustration:



3. Click Button 1 on a keyword to select it and choose the **Display** button *or* double-click Button 1 on the keyword to go to the topic associated with the keyword. If multiple topics exist, they are displayed in the **Topics Found** dialog box, as shown in the following illustration:



4. Select a topic name from the list and choose the **Display** button or double-click Button 1 on a topic name to display the topic.

Session 1: Start a MAX+PLUS II Session

In this session, you will start MAX+PLUS II to begin creating your project.

This tutorial assumes that the MAX+PLUS II working directory, which has the default name **\max2work**, appears on the **d:** drive on your computer. If you installed the MAX+PLUS II working directory in a different drive and/or directory, substitute the appropriate drive and/or directory name.

To start MAX+PLUS II:

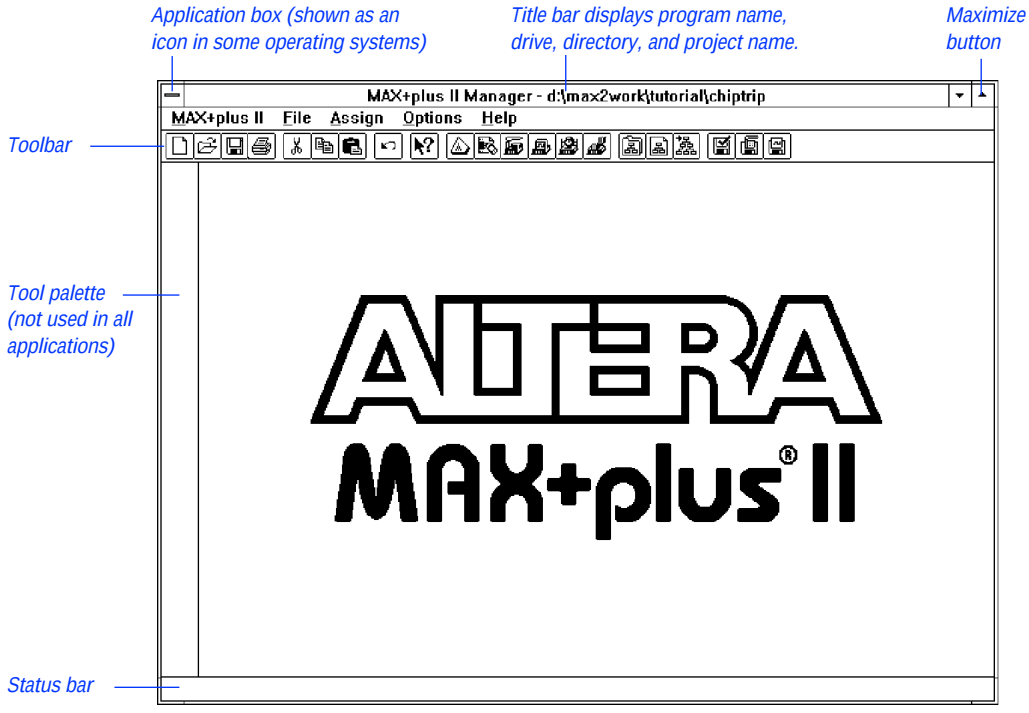
1. Double-click Button 1 on the MAX+PLUS II icon. On a PC running Windows, this icon appears in the MAX+PLUS II program group in the Program Manager window.

   or:

   Type `maxplus2` ↵ at the command line.

   The MAX+PLUS II Manager window opens. The title bar displays the name of the program (MAX+PLUS II), and a drive and directory name (**d:\max2work\tutorial**). The current project name, **chiptrip**, is appended to the drive and directory names.

See the following illustration:



All MAX+PLUS II applications contain a toolbar and status bar which you can turn on and off using the **Preferences** command (Options menu).

2. To turn the toolbar and status bar on and off:

   a. Choose **Preferences** from the Options menu. The **Preferences** dialog box is displayed.

   b. Turn the *Show Toolbar* and/or *Show Status Bar* options on or off by clicking Button 1 on their checkboxes.

   c. Choose **OK**.

The toolbar displays buttons and drop-down list boxes that provide quick access to frequently used commands. (Additional buttons are available on the tool palette in some applications.) The status bar provides a brief description of a highlighted menu command or an item in the toolbar or tool palette when you move the mouse pointer over it.

In some MAX+PLUS II applications, items on the far right of the toolbar are unavailable if your monitor is set to VGA display mode. All toolbar buttons and drop-down lists are available in larger screen displays.

If you wish, you can switch to an alternate combination of toolbar buttons for VGA displays. Go to "Setting MAX+PLUS II Preferences" using **Search for Help on** (Help menu) for instructions.

3. Maximize the MAX+PLUS II window by clicking Button 1 on the **Maximize** button, as shown in the illustration on page 166.

If you wish to exit MAX+PLUS II, choose **Exit MAX+PLUS II** (File menu) or double-click Button 1 on the application icon (or box) in the top left corner of the MAX+PLUS II window, as shown in the illustration on page 166.

Session 2: Create a Graphic Design File

In this session, you will specify the project name and use the MAX+PLUS II Graphic Editor to enter and save **tick_cnt.gdf**, which counts the number of speeding tickets you collect during your trip. This session includes the following steps:

1. Create a new file.
2. Specify the project name.
3. Select a palette tool.
4. Enter logic function symbols.
5. Set and show guidelines.
6. Move a symbol.
7. Enter input and output pins.
8. Name the pins.
9. Connect the symbols.
10. Connect nodes and buses by name.
11. Save the file and check for basic errors.
12. Create a default symbol.
13. Close the file.

Remember, all files for the **chiptrip** tutorial are available in the **\max2work\chiptrip** subdirectory.

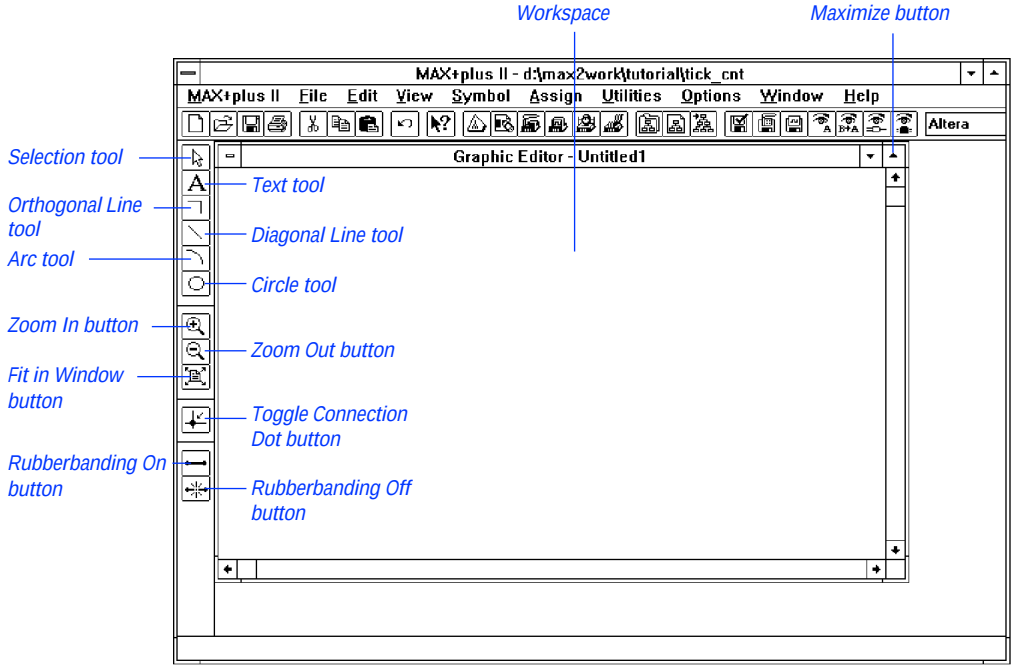
1. Create a New File

In this step, you will create a new GDF called **tick_cnt.gdf**.

To create the file:

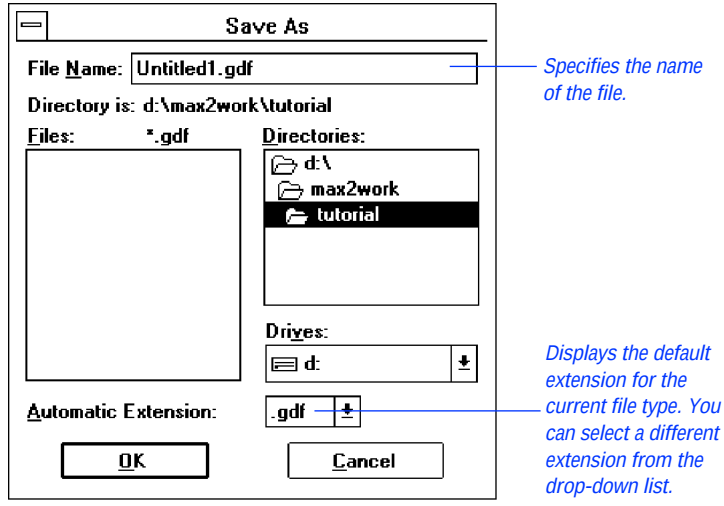
1. Choose **New** (File menu). The **New** dialog box is displayed.
2. Select *Graphic Editor file*.
3. Select the *.gdf* filename extension in the drop-down list box.
4. Choose **OK**.

   An untitled Graphic Editor window opens, as shown in the following illustration:



5. If necessary, maximize the Graphic Editor window by clicking Button 1 on the **Maximize** button in the Graphic Editor title bar.

6. To save the file, choose **Save As** (File menu). The **Save As** dialog box is displayed, as shown in the following illustration:



7. Type `tick_cnt.gdf` in the *File Name* box.

8. If **\max2work\tutorial** does not appear in the *Directory is* field as the current directory, select it in the *Directories* box.

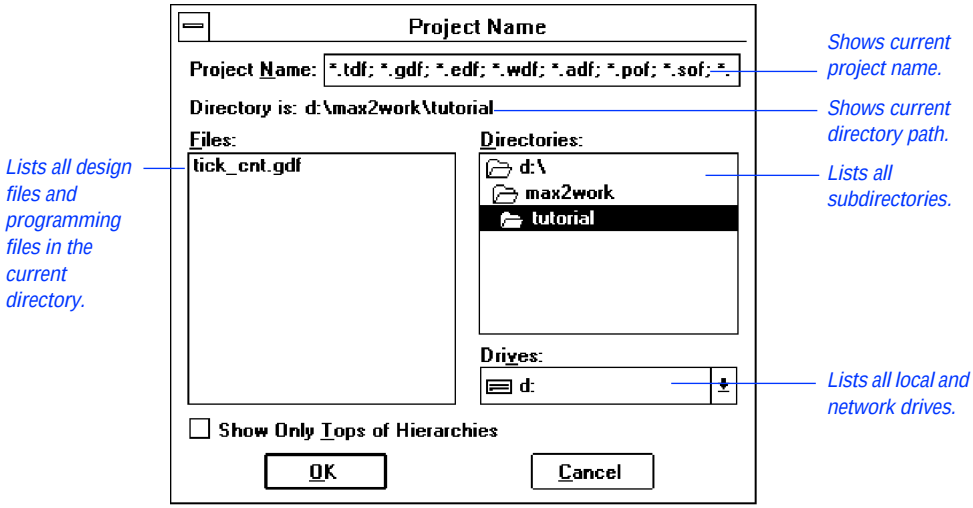
9. Choose **OK** to save the **tick_cnt.gdf** file.

2. Specify the Project Name

In MAX+PLUS II, you must specify a design file as your current project before you can compile it or perform any other batch processing such as simulation. MAX+PLUS II processes one project at a time, and you must ensure that all design files in a project appear in that project's hierarchy. You should always create a separate subdirectory for each new project. When you enter the project name, you also specify the name of the subdirectory where the project will be stored. If the subdirectory does not exist, MAX+PLUS II can create it for you.

To specify the project name:

1. Choose **Project Name** (File menu). The **Project Name** dialog box is displayed:



2. If necessary, turn off the *Show Only Tops of Hierarchies* option.

3. Select **tick_cnt.gdf** in the *Files* box.

4. Choose **OK**.

The MAX+PLUS II title bar changes to display the new project name:

```
MAX+plus II Manager - d:\max2work\tutorial\tick_cnt
```

As an alternative to using the **Project Name** command, you can simply choose the **Project Set Project to Current File** command (File menu) when **tick_cnt.gdf** is open in the active Graphic Editor window.

3. Select a Palette Tool

In this step, you will select from the various palette tools available for the Graphic Editor. In the Graphic, Symbol, and Waveform Editors, the pointer changes shape, depending upon the current selected palette tool and the object under the mouse pointer. The Selection tool, which has an arrow-shaped pointer, is the default palette tool when you first open a Graphic Editor window. As an exercise, you will select the Orthogonal Line tool from the tool palette.

To select the Orthogonal Line tool from the palette:

Click Button 1 on the Orthogonal Line tool, as shown in the illustration on page 169.

The pointer changes into a +-shaped pointer if you select the Orthogonal, Diagonal, Arc, or Circle tool. If you select the Text tool, the pointer is an inverted "t" that changes into an I-shaped pointer when it passes over editable text.

The Selection tool is a "smart" tool. When this tool is selected, the arrow-shaped Selection pointer automatically changes into the Orthogonal Line drawing or Text Editing pointer when it passes over different objects in the Graphic Editor window:

- When the Selection pointer passes over the end of a line, a connection dot, or a symbol pinstub, it changes into the +-shaped Orthogonal Line drawing pointer, which allows you to draw lines and enter or delete connection dots. This "smart" behavior means that you need to select a line drawing tool from the tool palette only if you wish to draw lines in empty space.

- When you double-click Button 1 on editable text, the Selection pointer changes into the I-shaped Text Editing pointer, which allows you to edit pin and node names, pin default values, and comments. This "smart" behavior means that you need to select the Text tool from the tool palette only if you wish to enter text in empty space.

- When the pointer passes over empty space, over the middle of a line or symbol, or over text, the Selection pointer has normal selection behavior, which allows you to select, move, and copy objects in the window.

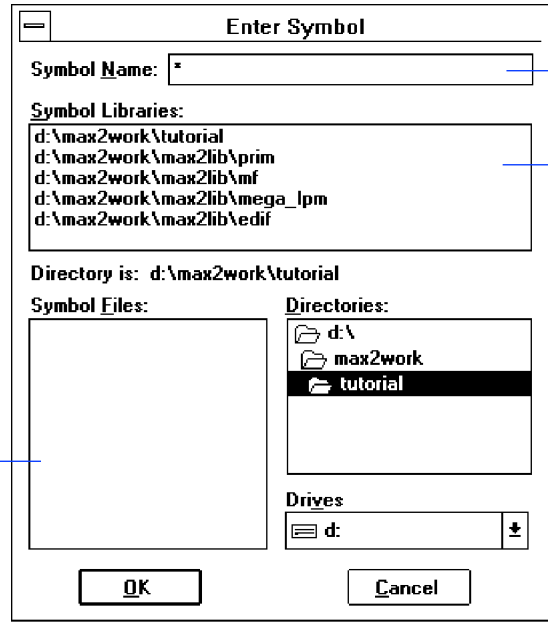
4. Enter Logic Function Symbols

MAX+PLUS II provides symbols for a variety of logic functions—including primitive, megafunction, and macrofunction symbols—that you can use in your Graphic Editor files.

To enter a symbol:

1. With the Selection pointer, click Button 1 in empty space in the Graphic Editor window to define an insertion point and choose **Enter Symbol** (Symbol menu).

The **Enter Symbol** dialog box is displayed:



Specifies the name of a symbol to be entered in your file.

Double-clicking Button 1 on a symbol library directory lists its contents in the Symbol Files box. MAX+PLUS II automatically searches each listed directory for a symbol when you type a symbol name in the Symbol Name box and choose OK.

Lists all symbols in the current directory.

SHORTCUTS

Double-clicking Button 1 in a blank space in the Graphic Editor window is a shortcut for this step. This action both defines an insertion point and opens the **Enter Symbol** dialog box.

Command shortcuts are described in more detail in Sessions 5 and 9 of this tutorial.

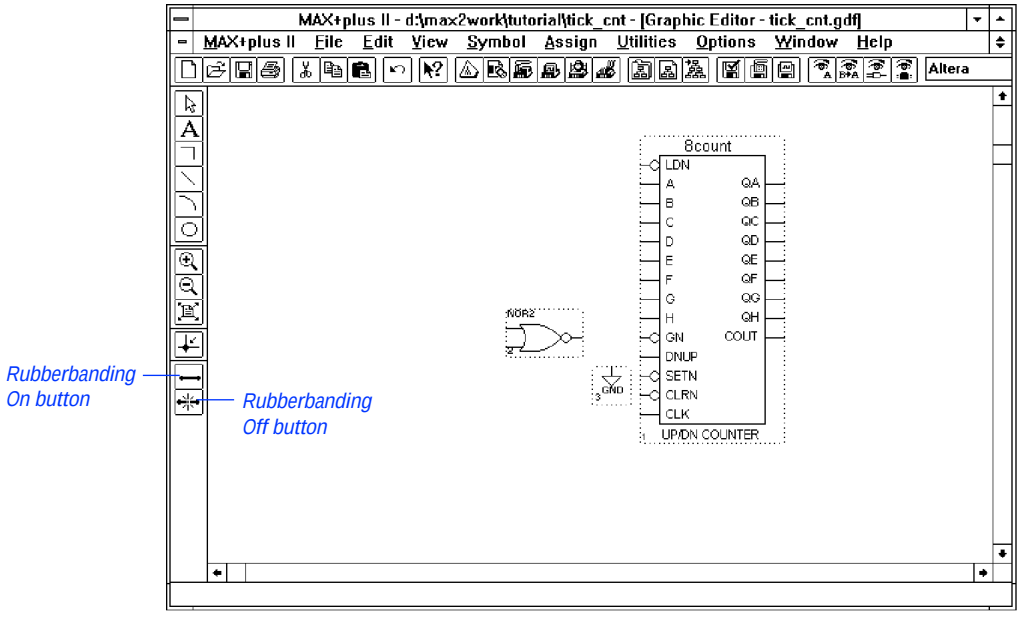
2. Type `8count` in the *Symbol Name* box.

3. Choose **OK**. The `8count` symbol is entered with its top left corner at the insertion point. The `8count` macrofunction is an 8-bit up/down binary counter. In **tick_cnt.gdf**, four bits of the `8count` macrofunction will be used to count the number of tickets received by the driver.

4. Repeat steps 1 through 3 to enter the `NOR2` and `GND` primitives to the left of the `8count` symbol.

MAX+PLUS II documentation conventions use all capital letters for primitive names. However, you should type all primitive, megafunction, and macrofunction names with lowercase letters in the **Enter Symbol** dialog box.

See the following illustration:



If you enter or move two symbols so that their borders and pinstubs touch, the symbols become logically connected. If you then move one of the symbols when **Rubberbanding** (Options menu) is turned on, a new node or bus line forms automatically between the connected pinstubs of the two symbols.

Choose from the toolbar and click Button 1 on the `8count`, `NOR2`, or `GND` symbols for information on each. On-line help provides complete information on all Altera-provided primitives, megafunctions, and macrofunctions.

You can also get help on megafunctions, primitives, and macrofunctions by choosing **Megafunctions/LPM**, **Primitives**, or **Old-Style Macrofunctions**, respectively, from the Help menu.

5. Set & Show Guidelines

To increase the readability of your schematic, you can align symbols to horizontal and vertical guidelines. You can specify the guideline spacing and display or hide the guidelines.

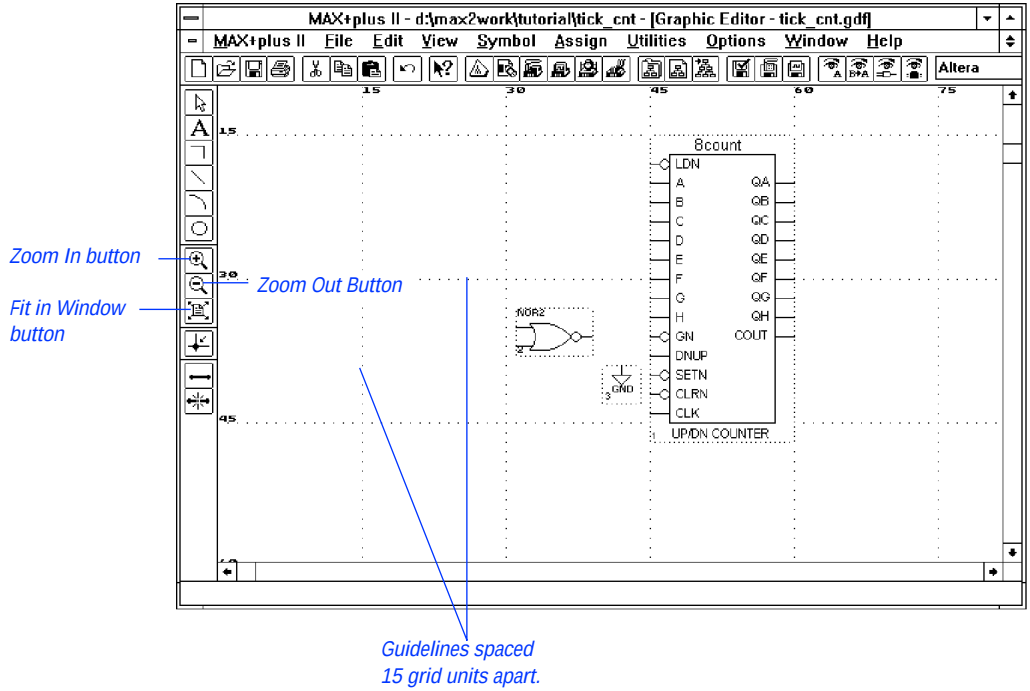
To set the spacing and show the guidelines:

1. Choose **Guideline Spacing** from the Options menu. The **Guideline Spacing** dialog box is displayed.

2. Type **15** in both the *X (Horizontal) Spacing* and *Y (Vertical) Spacing* boxes to specify 15 grid-unit spacing between guidelines.

3. Choose **OK**.

4. Turn on **Show Guidelines** (Options menu). When the command is turned on, a checkmark appears next to the command name in the menu. The guidelines appear as shown in the following illustration:



As you edit your schematic, you can change the window display to view larger or smaller portions of the file by choosing the **Zoom In**, **Zoom Out**, and **Fit in Window** buttons on the tool palette, as shown in the previous illustration.

6. Move a Symbol

To move and align the `8count` symbol:

1. With the Selection pointer, press or click Button 1 on `8count` to select the symbol.
2. While pressing Button 1, drag the symbol and position the top left corner of `8count` at the closest guideline intersection. An outline of the symbol moves with the pointer so that you can position it accurately.
3. Once the symbol is in position, release Button 1.

You can move any symbol, graphic, or block of text that can be selected with Button 1 in the MAX+PLUS II Graphic or Symbol Editor by dragging it with the Selection pointer.

Go to "Moving an Object" and "Selecting an Object" using **Search for Help on** (Help menu).

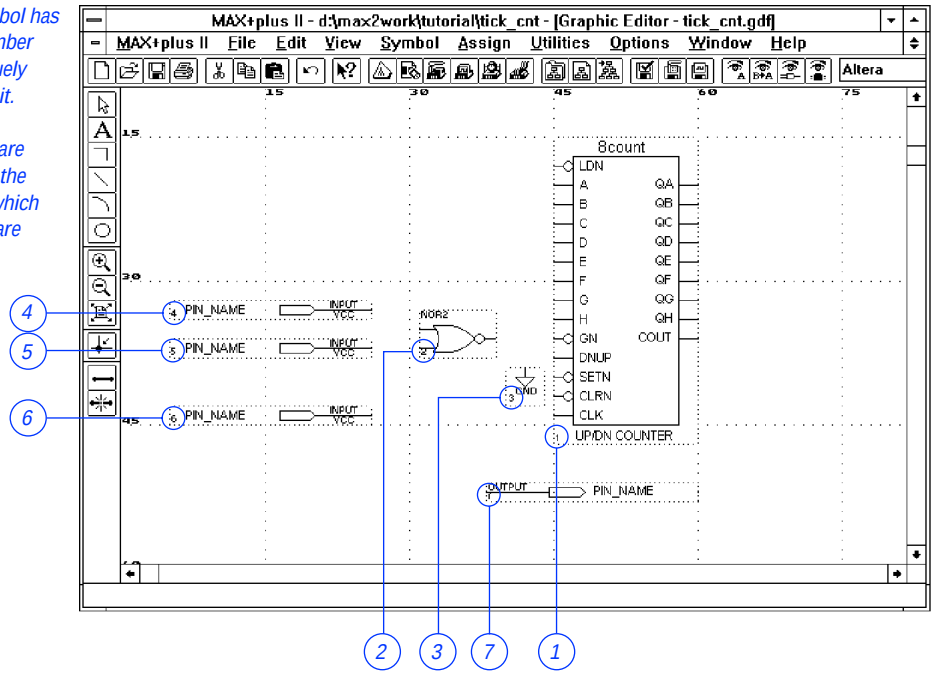
7. Enter Input & Output Pins

To enter the `INPUT` and `OUTPUT` pins:

1. With the Selection pointer, double-click Button 1 in an empty space to the left of the `8count` symbol to open the **Enter Symbol** dialog box, type `input` in the *Symbol Name* box, and choose **OK**. The `INPUT` pin symbol is displayed.
2. Press Ctrl and then press Button 1 on the `INPUT` symbol. While holding Ctrl and Button 1 down, drag the mouse down to create a copy of the symbol and place it below the original. (The symbol is copied but not placed on the Clipboard.)
3. Repeat step 2 to create the third `INPUT` symbol.
4. Repeat step 1 to enter the `OUTPUT` symbol below the `8count` symbol.

See the following illustration:

Each symbol has an ID number that uniquely identifies it. These ID numbers are based on the order in which symbols are entered.



A symbol identification number is located at the bottom left corner of each symbol. It is automatically assigned based on the order in which you enter the symbols (i.e., the first symbol entered is assigned the ID number 1). It uniquely identifies each instance of a symbol within the GDF. The symbol ID numbers in your file may differ from those in the illustrations, depending on the order in which you enter the symbols. However, these differences will not cause any errors.

8. Name the Pins

You will now name the input and output pins. Symbols in this procedure are identified as <*symbol name*>:<*symbol ID*>, e.g., `INPUT:4` is the `INPUT` symbol with the symbol ID number `4`, as entered in the previous step.

If you did not enter the symbols in the described sequence, your symbol ID numbers will differ from those in the illustration. These differences will not cause any errors.

To assign a pin name:

1. With the Selection pointer, double-click Button 1 on the default pin name "PIN_NAME" of INPUT:4 to select the entire name.

2. Type get_ticket1. The new name replaces the default pin name. The get_ticket1 signal will be used to enable the counter. When it goes high, the count value will increment by one.

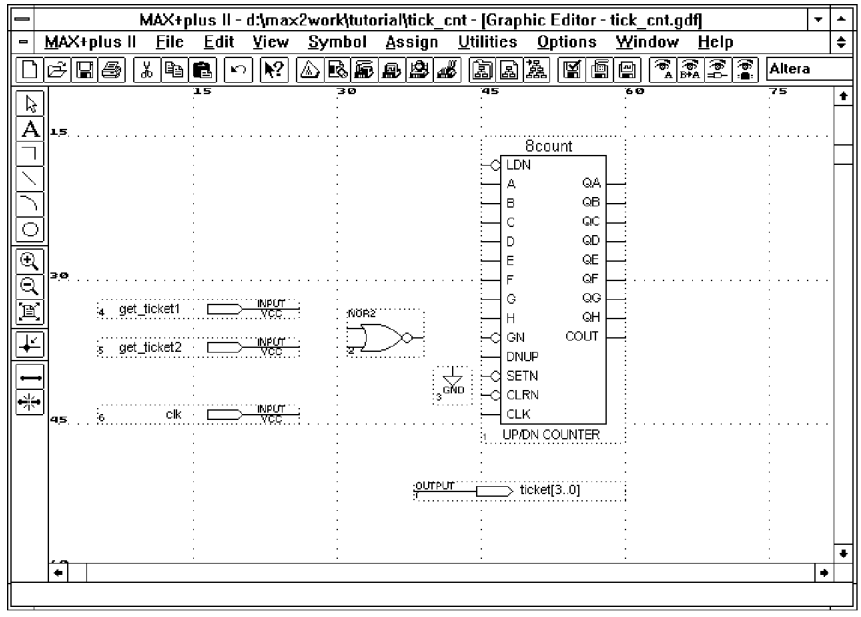
   

   You can also use the Text tool to edit and enter text. Double-click Button 1 to select the entire name, or drag Button 1 to select a portion of the name you wish to edit.

3. Rename the remaining INPUT and OUTPUT pins as shown in the following list. If you press ↵ after you edit a pin name, the next pin name below it is automatically selected for editing.

Primitive:	Name:	Description:
INPUT:5	get_ticket2	Same as get_ticket1.
INPUT:6	clk	This signal is the Clock for the **tick_cnt.gdf** counter.
OUTPUT:7	ticket[3..0]	These signals represent the counter bit outputs. The name is given in single-range bus name format, which is used to create an array of four output pins. In this case, ticket[3..0] is the bus that represents the pins ticket3, ticket2, ticket1, and ticket0.

See the following illustration:



Go to "Pin & Node Names" and "Bus Names" using **Search for Help on** (Help menu).

9. Connect the Symbols

You can use the "smart" Selection tool to draw most of the lines you need to connect symbols in a Graphic Editor file—the Selection pointer automatically turns into the Orthogonal Line drawing pointer when it is over a pinstub or connection dot, or at the end of a line. You can also use the Orthogonal and Diagonal Line tools to connect symbols.

To draw a line:

1. With the Selection pointer, move symbols so they line up with the appropriate pinstubs or other symbols, as shown in the preceding illustration.

2. Choose the solid line style from the top of the **Line Style** submenu (Options menu). This line style is the recommended line style for nodes.

3. Point to the output pinstub of the `get_ticket1` input pin. The Selection pointer turns into the +-shaped Orthogonal Line drawing pointer when it is close to the pinstub.

4. Press Button 1 to define the start of a line.

5. While pressing Button 1, drag the mouse to draw a line that connects to the uppermost input pinstub on the `NOR2` gate.

6. Release Button 1.

With the Orthogonal Line drawing pointer, you can draw straight lines or lines with a single bend. If you cannot draw a straight line or a line with a single bend to connect two symbols, you must draw two lines to create the two bends needed to make the connection. After you draw the first line, draw a second line that connects to the end of the first. Lines merge automatically if they have the same line style.

If you need to delete a line, click Button 1 on the line to select it and press the Del or Backspace key.

7. Repeat steps 3 through 7 to connect the remaining symbols as shown in the following table and in the illustration on page 181:

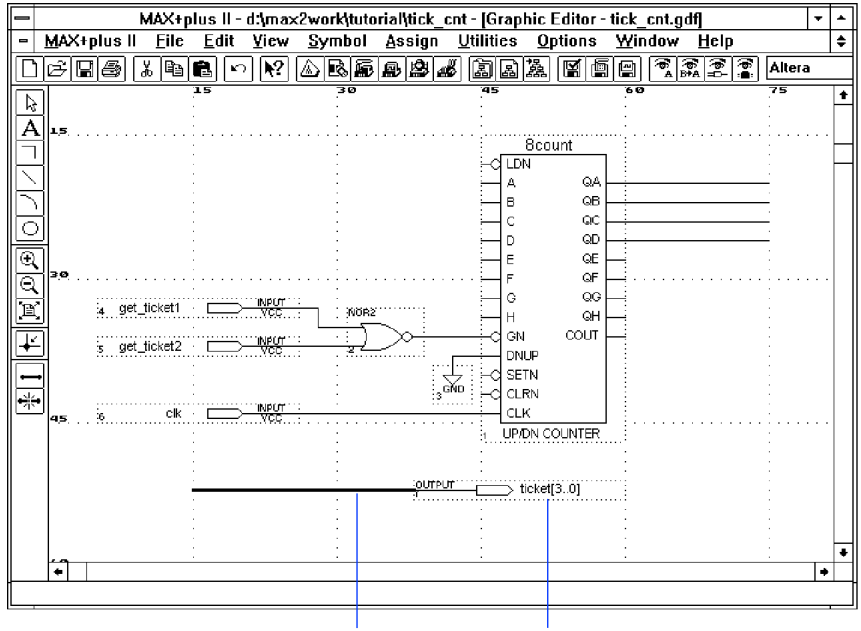
Draw Line From:	**To:**
`INPUT` pin `get_ticket2`	`NOR2` input
Output pinstub of `NOR2`	`GN` input of `8count`
`DNUP` input of `8count`	`GND`
`INPUT` pin `clk`	`CLK` input of `8count`
`QA` output of `8count`	Next vertical guideline on the right (i.e., draw line but do not physically connect it to another symbol)
`QB` output of `8count`	Same as `QA`
`QC` output of `8count`	Same as `QA`
`QD` output of `8count`	Same as `QA`
`OUTPUT` pin `ticket[3..0]`	Next vertical guideline on the left, as with `QA`

The line that extends from the `ticket[3..0]` output pin should be a bus line. To change the line into a bus line:

1. Point to the line that extends from the output pin named `ticket[3..0]` and click Button 1 to select the line.

2. Choose the bus line style, i.e., the thick line style, from the **Line Style** submenu (Options menu).

See the following illustration:



Choose from the toolbar and click Button 1 on the bus line to go to "Buses" in MAX+PLUS II Help.

10. Connect Nodes & Buses by Name

You can connect the nodes (i.e., lines) that are attached to the output pinstubs QA, QB, QC, and QD on the 8count symbol by name to the bus line that is attached to the input of ticket[3..0]. Nodes and buses are connected by assigning appropriate names to them; they need not be physically connected. A logical connection exists only if each member of a bus has the same name as one of the nodes. For example, to connect bus b[1..0] (with members b1 and b0) to two nodes that are not physically connected to the bus, you must name the nodes b1 and b0.

To assign names to the nodes and bus:

1. Change the text size and font to 10-point Arial font:

   a. Choose **Text Size** (Options menu). If **10** is not checked on the submenu, select it from the list of available text sizes.

   b. Choose **Font** (Options menu). If **Arial** is not checked on the submenu, select it from the list of available fonts.

2. With the Selection pointer, select the node that extends from the QA pinstub of the 8count symbol by clicking Button 1 on the line. A small square insertion point appears below the line to show the insertion point for the name.

3. Type ticket0. The name appears above the line.

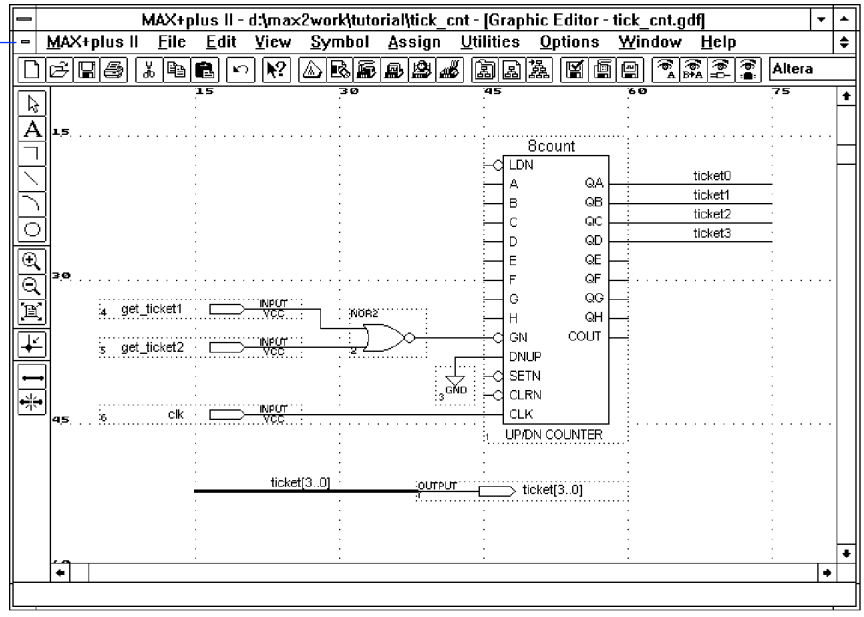
   If a name overlaps a symbol, you can move it to another location on the node or bus line by simply dragging it to a new position with Button 1 in the same way that you would move a symbol. See "6. Move a Symbol" on page 176 for more information.

4. Repeat steps 1 through 3 for the remaining nodes and the bus:

Pinstub/Pin Name:	Node/Bus Name:
QB	ticket1
QC	ticket2
QD	ticket3
ticket[3..0]	ticket[3..0]

See the following illustration:

Document box (shown as an icon in some operating systems)



Once you have assigned these names, the nodes `ticket3`, `ticket2`, `ticket1`, and `ticket0` are automatically connected to the bus `ticket[3..0]` by name, even though they are not physically connected.

11. Save the File & Check for Basic Errors

To ensure that you have entered the logic correctly, you can save the file and check for simple errors.

To save the file and check for errors:

1. Choose **Project Save & Check** from the File menu. The file is saved and the MAX+PLUS II Compiler window opens; the Compiler Netlist Extractor module checks the file for errors, updates the Hierarchy Display, and displays a message indicating the number of errors and warnings.

2. If **Project Save & Check** is successful and there are no errors or warnings, choose **OK** to close the message box.

If the Compiler issues any error or warning messages and the Message Processor window is not automatically displayed, you can open it by choosing **Message Processor** (MAX+PLUS II menu). Select a message in the Message Processor window and choose **Locate** to find its source(s) or choose **Help on Message** to get an explanation. See "10. Locate the Source of a Message" on page 226 for more information. You should correct any errors in your design file, and save and check it until it is error-free.

3. Double-click Button 1 on the document icon (or box), as shown in the illustration on page 183, to close the Compiler window and return to the Graphic Editor.

12. Create a Default Symbol

You will now create a Symbol File (**.sym**) that represents the current file. The symbol in this file can be used in any other Graphic Design File (**.gdf**).

To create a default symbol:

✓ Choose **Create Default Symbol** (File menu). If a symbol for a file already exists, MAX+PLUS II asks you whether it is OK to overwrite the existing symbol. Choose **OK** to ensure that you have the most up-to-date information in your Symbol File.

13. Close the File

To close the **tick_cnt.gdf** file:

✓ Choose **Close** from the File menu or double-click Button 1 on the document icon (or box), as shown in the illustration on page 183. The Graphic Editor window displaying the **tick_cnt.gdf** file closes automatically.

Session 3: Create Two Text Design Files

In this session, you will use the MAX+PLUS II Text Editor to enter and save two Text Design Files (**.tdf**) written in the Altera Hardware Description Language (AHDL). The first TDF, **time_cnt.tdf**, measures the time it takes for your vehicle to reach Altera by counting Clock pulses. The **auto_max.tdf** file describes the functions of the `street_map` state machine, and determines the direction and acceleration of your automobile. This session includes the following steps:

1. Create a new file and specify the project name.
2. Turn on syntax coloring.
3. Enter the design name, inputs, and outputs.
4. Declare a register.
5. Enter Boolean equations.
6. Enter an If Then Statement.
7. Check for syntax errors and create a default symbol.
8. Copy **auto_max.tdf** and create a default symbol.

Go to MAX+PLUS II AHDL Help or the ***MAX+PLUS II AHDL*** manual for more detailed information on AHDL. The on-line help and manual contain the same information at the time of printing, but the on-line information is always the most up-to-date. On-line help also has links to related help topics, and pop-up examples and glossary definitions to help you find the information you need as quickly as possible. You should use the manual as a take-home reference and use on-line help when you are at your computer.

1. Create a New File & Specify the Project Name

In this step, you will create a new TDF called **time_cnt.tdf** in AHDL, and name the project.

To specify the project name and create a new file:

1. Choose **New** (File menu), select *Text Editor file*, and choose **OK** to open an untitled Text Editor window (see "1. Create a New File" on page 168).

2. If necessary, maximize the Text Editor window by clicking Button 1 on the **Maximize** button in the top right corner of the Text Editor title bar.

3. Choose **Save As** (File menu). Type `time_cnt.tdf` in the *File Name* box.

4. Make sure that **\max2work\tutorial** appears as the current directory in the *Directory is* field. Choose **OK** to save the **time_cnt.tdf** file.

5. Choose **Project Set Project to Current File** or **Project Name** (File menu) and change the project name to **time_cnt** (see "2. Specify the Project Name" on page 170).

2. Turn on Syntax Coloring

To make sections of your TDF more visible, you can use the Text Editor's **Syntax Coloring** command. Syntax coloring allows you to see the different parts of a TDF in different colors. For example, keywords are displayed in one color and comments are displayed in another, making them easier to distinguish on screen.

To turn on syntax coloring:

✓ Choose **Syntax Coloring** (Options menu). When the command is turned on, a checkmark appears next to the command name in the menu.

3. Enter the Design Name, Inputs & Outputs

In this step, you will start entering the AHDL file by creating a Subdesign Section that declares the input and output ports of the file.

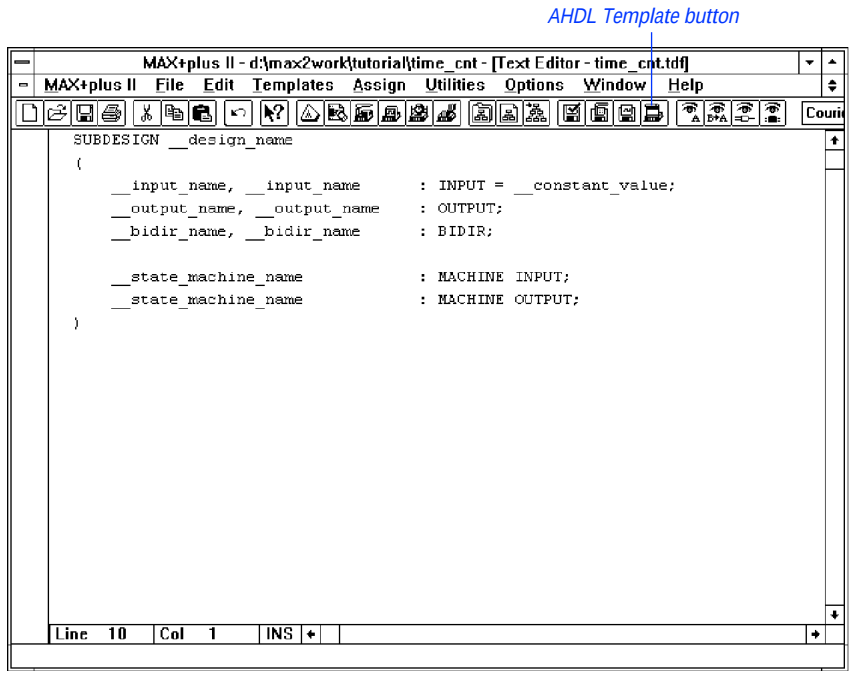
This tutorial follows the guidelines for indentation and white space provided in the "AHDL & VHDL Style Guide" in MAX+PLUS II AHDL Help.

To enter the design name, inputs, and outputs:

1. Choose **AHDL Template** (Templates menu). The AHDL Template dialog box is displayed.

2. Select *Subdesign Section* in the *Template Section* box.

3. Choose **OK**. A template for the Subdesign Section appears at the insertion point. Each variable name starts with two underscores (__), and each keyword is capitalized, as shown in the following illustration:



SHORTCUTS

Shortcuts for opening the **AHDL Template** dialog box:

✓ Press Button 2 and choose **AHDL Template** from the pop-up menu.

or:

✓ Click Button 1 on the **AHDL Template** toolbar button at the top of the window, as shown in the previous illustration.

☞ To improve readability, you can change the font and/or text size that appears in the Text Editor window with the **Font** and/or **Text Size** commands (Options menu). This tutorial shows files entered with 10 point Courier New font text.

4. Double-click Button 1 on the `__design_name` variable and type `time_cnt`.

5. To add the names of the inputs, double-click Button 1 on the first `__input_name` variable and type `enable`, then double-click Button 1 on the second `__input_name` variable and type `clk`. Delete the `__constant_value` variable and the equal sign (=) preceding it.

6. To add the names of the outputs, double-click Button 1 on the first `__output_name` variable and type `time[7..0]`. Delete the second `__output_name` variable and the comma (`,`) preceding it.

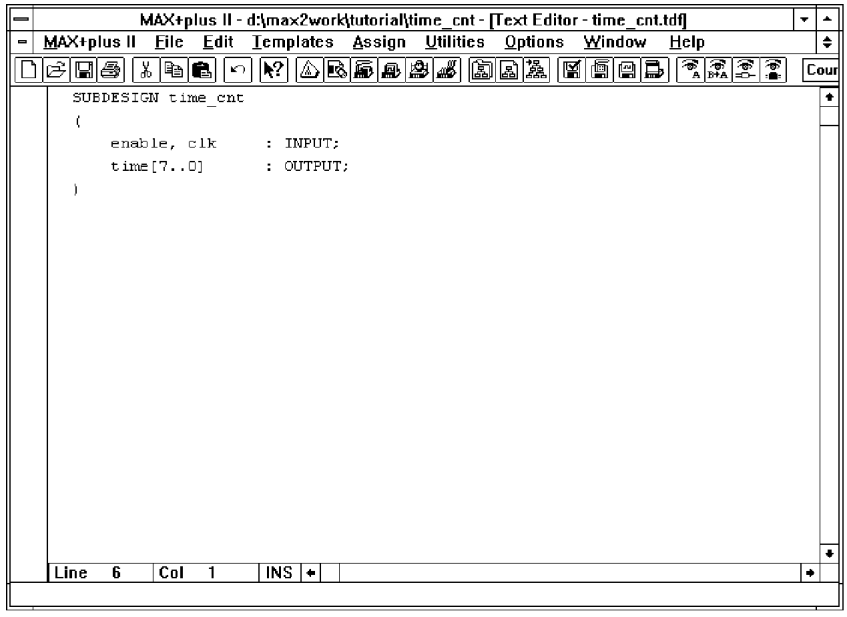
7. Delete the lines containing the `BIDIR`, `MACHINE INPUT`, and `MACHINE OUTPUT` keywords.

8. Add spaces and/or tabs to improve readability.

☞ To indent text easily, you can turn on **Auto-Indent** (Options menu) before typing text, or select existing text to indent and choose **Increase Indent** (Edit menu). You can set the tab spacing with **Tab Stops** (Options menu). To delete any unnecessary tabs, choose **Decrease Indent** (Edit menu) or use the Del or Backspace key.

Choose from the toolbar, then choose a menu command for more information on these commands.

See the following illustration:



The ports `enable` and `clk` are inputs, and the ports `time[7..0]` are outputs, of **time_cnt.tdf**.

Choose  from the toolbar and click Button 1 on the `SUBDESIGN` keyword in the file to go to "Subdesign Section" in MAX+PLUS II AHDL Help.

4. Declare a Register

You can create a register with a Register Declaration in the Variable Section. In this example, you will declare eight instances of a D flipflop with the names `count[7..0]`.

To declare the registers:

1. On a new line after the Subdesign Section, type the `VARIABLE` keyword and press ↵.
2. Choose **AHDL Template** (Templates menu).
3. Select *Register Declaration* in the *Template Section* box.

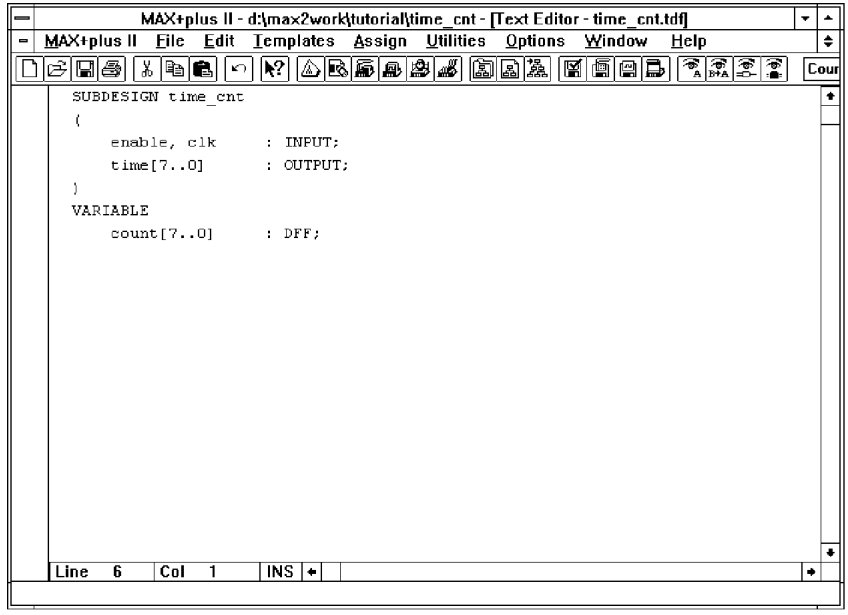
4. Choose **OK**. A template for the Register Declaration appears at the insertion point.

5. While the template text is still selected, choose **Increase Indent** (Edit menu) to indent the text to the right.

6. Double-click Button 1 on the `__register_instance_name` variable and type `count[7..0]`.

7. Double-click Button 1 on the `__register_name` variable and type `DFF`.

See the following illustration:



Go to "Declaring Registers (AHDL)" using **Search for Help on** (Help menu).

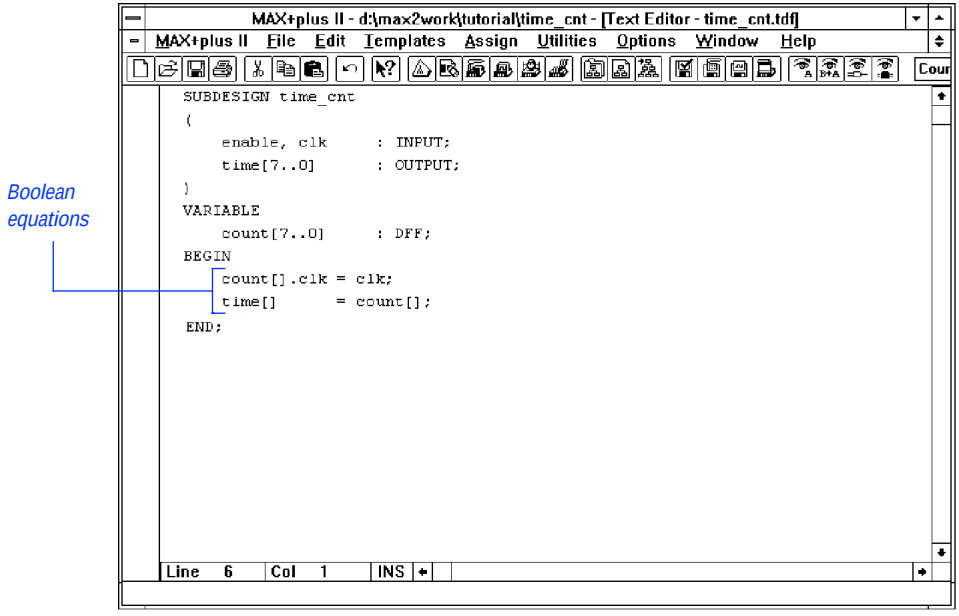
5. Enter Boolean Equations

Boolean equations can be used to connect signals to data ports in a D flipflop. Boolean equations are entered in the Logic Section, which is delimited by the `BEGIN` and `END` keywords.

1. Place the cursor after `DFF;` and press ↵.
2. Type `BEGIN` and press ↵.
3. Type `count[].clk = clk;` and press ↵.
4. Type `time[] = count[];` and press ↵.
5. Type `END;`.

To indent the equations, you can enter tab characters or select an equation and choose the **Increase Indent** command (Edit menu). To delete any unnecessary tabs, choose **Decrease Indent** (Edit menu) or use the Del or Backspace key.

See the following illustration:



Once you have defined the group, square brackets (`[]`) are a shorthand way of specifying the entire range. The first equation connects the subdesign's `clk` input to the flipflops' Clock ports. The second equation sets the `time[]` inputs equal to the `count[]` variables.

Go to "Boolean Equations (AHDL)" using **Search for Help on** (Help menu).

6. Enter an If Then Statement

In this step, you will define the logic for the design with an If Then Statement:

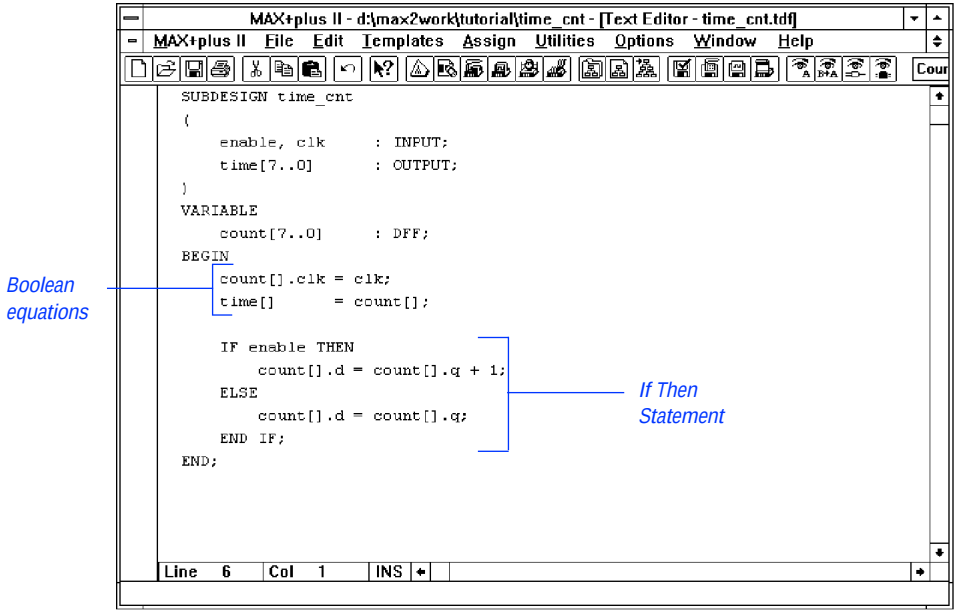
1. Place the cursor after `count[];` and press ↵ twice to add white space for readability.

2. If necessary, press Tab to align the cursor with the Boolean equations, then choose **AHDL Template** (Templates menu).

3. Select *If Then Statement* in the *Template Section* box.

4. Choose **OK**. A template for the If Then Statement appears at the insertion point.

5. Edit the variables to create the If Then Statement shown in the following illustration:



If the `enable` signal is high, the If Then Statement assigns the value `count[].q + 1` to `count[].d`; if `enable` is low, `count[].d` remains unchanged.

Go to "Implementing Conditional Logic (AHDL)" using **Search for Help on** (Help menu).

Choose from the toolbar and click Button 1 on the `THEN` keyword in the file to go to "If Then Statement" in MAX+PLUS II AHDL Help.

7. Check for Syntax Errors & Create a Default Symbol

You will now check the file for syntax errors to ensure that it was entered correctly, and then create a default symbol for use in the top-level GDF.

1. Choose **Project Save & Check** (File menu). See "11. Save the File & Check for Basic Errors" on page 183.

2. Go back to the **time_cnt.tdf** file by choosing its name from the bottom of the Window menu or by choosing **Text Editor** from the MAX+PLUS II menu.

3. Choose **Create Default Symbol** (File menu) to create the **time_cnt.sym** Symbol File. Double-click Button 1 on the Document Control Menu box to close the Compiler window.

4. Choose **Close** (File menu) to close the **time_cnt.tdf** window.

8. Copy auto_max.tdf & Create a Default Symbol

The **auto_max.tdf** file describes the functions of the `street_map` state machine. It determines the direction and acceleration of your automobile.

1. Enter the TDF as shown in Figure 3-3 *or* copy **auto_max.tdf** from the **\max2work\chiptrip** subdirectory into the **\max2work\tutorial** subdirectory.

   If you copy the file, set your tab stops to four spaces with **Tab Stops** (Options menu) to view the columns of text in proper alignment.

2. Change the project name to **auto_max**, save and check the file for errors, and create a default symbol for **auto_max.tdf** as described above in "7. Check for Syntax Errors & Create a Default Symbol."

3. Close **auto_max.tdf**.

Figure 3-3. auto_max.tdf (Part 1 of 2)

```
CONSTANT NORTH = B"00";   % Create descriptive names for numbers %
CONSTANT EAST  = B"01";   % for use elsewhere in file           %
CONSTANT WEST  = B"10";
CONSTANT SOUTH = B"11";

SUBDESIGN auto_max
(
  dir[1..0], accel, clk, reset          : INPUT;  % File inputs  %
  speed_too_fast, at_altera, get_ticket : OUTPUT; % File outputs %
)
VARIABLE
  street_map : MACHINE                % Create state machine with bits q2, %
               OF BITS (q2,q1,q0)     % q1 & q0 as outputs of register     %
               WITH STATES (
                 yc,                  % Your company             %
                 mpld,                % Marigold Park Lane Drive %
                 epld,                % East Pacific Lane Drive  %
                 gdf,                 % Great Delta Freeway      %
                 cnf,                 % Capitol North First      %
                 rpt,                 % Regal Park Terrace       %
                 epm,                 % East Pacific Main        %
                 altera );            % Your one-stop programmable logic shop %

BEGIN
  street_map.clk   = clk;   % input pin "clk" connects to state machine clk    %
  street_map.reset = reset; % input pin "reset" connects to state machine reset%
  % File outputs default to GND unless otherwise specified                     %

  TABLE                           % Define state transitions %

  % Present                         Next                    %
  % State       Inputs              State       Outputs     %

  street_map,dir[1..0],accel => street_map,get_ticket,at_altera,speed_too_fast;
  % ------------------------------------------------------------------------- %
  % When street_map is in the state yc, dir[1..0]=00, and accel=0, it enters  %
  % the state rpt and outputs 0 for get_ticket, at_altera, and speed_too_fast.%
    yc,         NORTH,     0     => rpt,        0,          0,         0;
    yc,         EAST,      0     => gdf,        0,          0,         0;
    yc,         NORTH,     1     => mpld,       0,          0,         1;
    yc,         EAST,      1     => cnf,        1,          0,         1;
    gdf,        NORTH,     0     => epld,       0,          0,         0;
    gdf,        WEST,      0     => yc,         0,          0,         0;
    gdf,        WEST,      1     => yc,         1,          0,         1;
```

Figure 3-3. auto_max.tdf (Part 2 of 2)

```
    gdf,        EAST,      0     => cnf,        0,          0,          0;
    gdf,        EAST,      1     => cnf,        1,          0,          1;
    gdf,        NORTH,     1     => mpld,       0,          0,          0;
    cnf,        NORTH,     0     => epm,        0,          0,          0;
    cnf,        WEST,      0     => gdf,        0,          0,          0;
    cnf,        NORTH,     1     => altera,     0,          0,          1;
    cnf,        WEST,      1     => yc,         1,          0,          1;
    rpt,        NORTH,     0     => mpld,       0,          0,          0;
    rpt,        NORTH,     1     => mpld,       0,          0,          1;
    rpt,        EAST,      0     => epld,       0,          0,          0;
    rpt,        EAST,      1     => epm,        0,          0,          1;
    rpt,        SOUTH,     0     => yc,         0,          0,          0;
    epld,       NORTH,     X     => mpld,       0,          0,          0;
    epld,       WEST,      0     => rpt,        0,          0,          0;
    epld,       WEST,      1     => rpt,        0,          0,          1;
    epld,       SOUTH,     X     => gdf,        0,          0,          0;
    epld,       EAST,      0     => epm,        0,          0,          0;
    epld,       EAST,      1     => epm,        0,          0,          1;
    epm,        NORTH,     0     => altera,     0,          0,          0;
    epm,        NORTH,     1     => altera,     0,          0,          1;
    epm,        SOUTH,     0     => cnf,        0,          0,          0;
    epm,        SOUTH,     1     => cnf,        0,          0,          1;
    epm,        WEST,      0     => epld,       0,          0,          0;
    epm,        WEST,      1     => rpt,        0,          0,          1;
    mpld,       WEST,      0     => rpt,        0,          0,          0;
    mpld,       SOUTH,     0     => epld,       0,          0,          0;
    mpld,       WEST,      1     => yc,         0,          0,          1;
    mpld,       SOUTH,     1     => gdf,        0,          0,          0;
    altera,     X,         X     => altera,     0,          1,          0;

    END TABLE;
END;
```

Session 4: Create a Waveform Design File

In this session, you will use the MAX+PLUS II Waveform Editor to enter and save a Waveform Design File (**.wdf**) called **speed_ch.wdf**. It includes the `speed` state machine, which contains the states `legal`, `warning`, and `ticket`. The **speed_ch.wdf** file checks whether or not your vehicle is traveling at a legal speed. The first time your car exceeds the speed limit, you are given a warning; the second time you speed, you get a speeding ticket. This session includes the following steps:

1. Create a new file and specify the project name.
2. Create input, output, and buried nodes.
3. Set the grid size and show the grid.
4. Edit the buried state machine node waveform.
5. Edit the input and output node waveforms.
6. Confirm the edits.
7. Check for basic errors and create a default symbol.

If you have not purchased the waveform design entry feature for MAX+PLUS II, you can use an AHDL TDF version of the **speed_ch.wdf** file, called **speed_ch.tdf**. This file is available in the **\max2work\chiptrip** subdirectory. Copy this file into your **\tutorial** subdirectory and proceed to "7. Check for Basic Errors & Create a Default Symbol" on page 209.

Go to "Creating a Waveform Design File" using **Search for Help on** (Help menu).

1. Create a New File & Specify the Project Name

In this step, you will create a new WDF called **speed_ch.wdf** and name the project.

To create a new file and specify the project name:

1. Choose **New** (File menu), select *Waveform Editor file*, select the *.wdf* extension in the drop-down list box, and choose **OK** to open an untitled Waveform Editor window. You must create the file with the **.wdf** extension to take advantage of design entry features in the Waveform Editor.

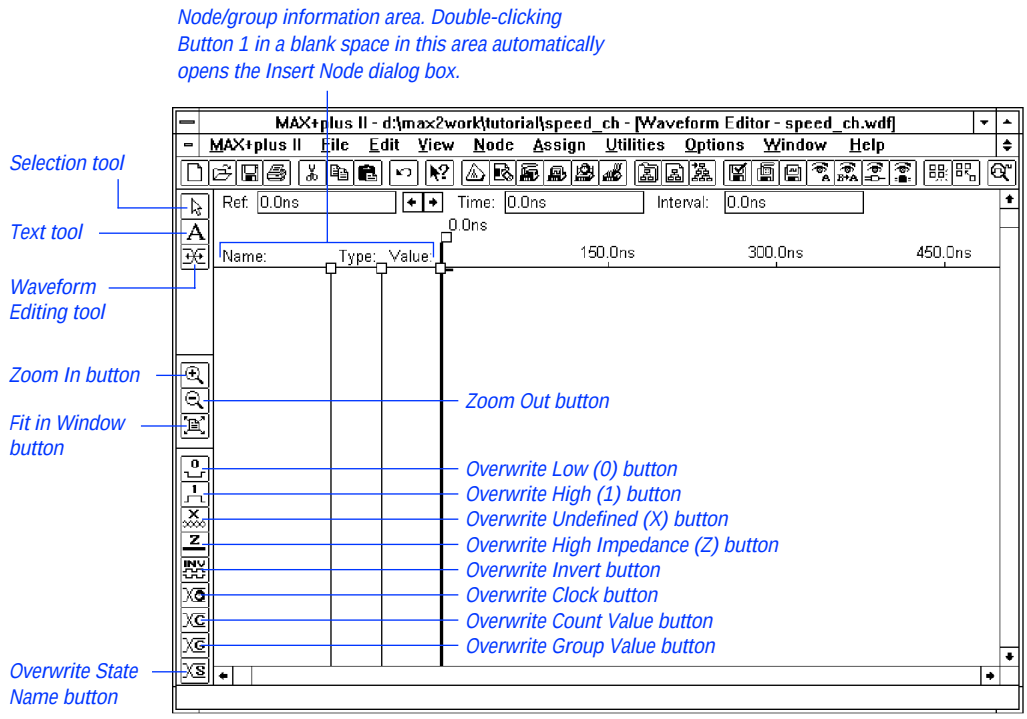
2. If necessary, maximize the Waveform Editor window by clicking Button 1 on the **Maximize** button in the title bar.

3. Choose **Save As** (File menu). Type `speed_ch.wdf` in the *File Name* box.

4. Make sure that **\max2work\tutorial** appears as the current directory in the *Directory is* field. Choose **OK** to save the **speed_ch.wdf** file.

5. Choose **Project Set Project to Current File** or **Project Name** (File menu) and change the project name to **speed_ch** (see "2. Specify the Project Name" on page 170).

See the following illustration:



2. Create Input, Output & Buried Nodes

You create a WDF by entering input node waveforms and the desired output node waveforms. You can also create optional buried node waveforms to provide logical links between inputs and outputs.

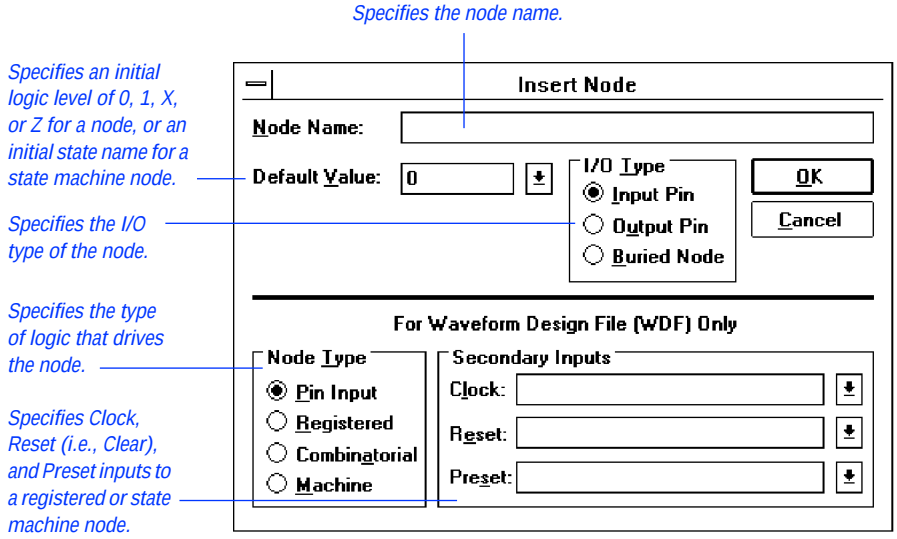
You will create three input nodes for **speed_ch.wdf**: `accel_in`, `reset`, and `clk`. The `accel_in` node represents your acceleration. When your car speeds up, the waveform's logic level changes from low (0) to high (1); when you slow down, the signal returns to low. The `reset` and `clk` nodes provide secondary inputs, i.e., Reset (Clear) and Clock inputs, to the registers that will be created to implement your buried and output nodes.

You will also create a buried node called `speed` that represents a state machine. This state machine informs you about possible states resulting from your level of speed.

Finally, you will create the `get_ticket` node that defines the output expected from your specified combination of inputs.

To create the nodes:

1. Click Button 1 in the topmost blank space in the node/group information area, as shown in the illustration on page 198, and choose **Insert Node** (Node menu). The **Insert Node** dialog box is displayed:



2. Type `accel_in` in the *Node Name* box.

3. Select *0* in the *Default Value* drop-down list box.

4. Select *Input Pin* under *I/O Type.*

5. In the bottom half of the dialog box, select *Pin Input* under *Node Type.*

6. Choose **OK**. The new node appears in the topmost blank space in the window.

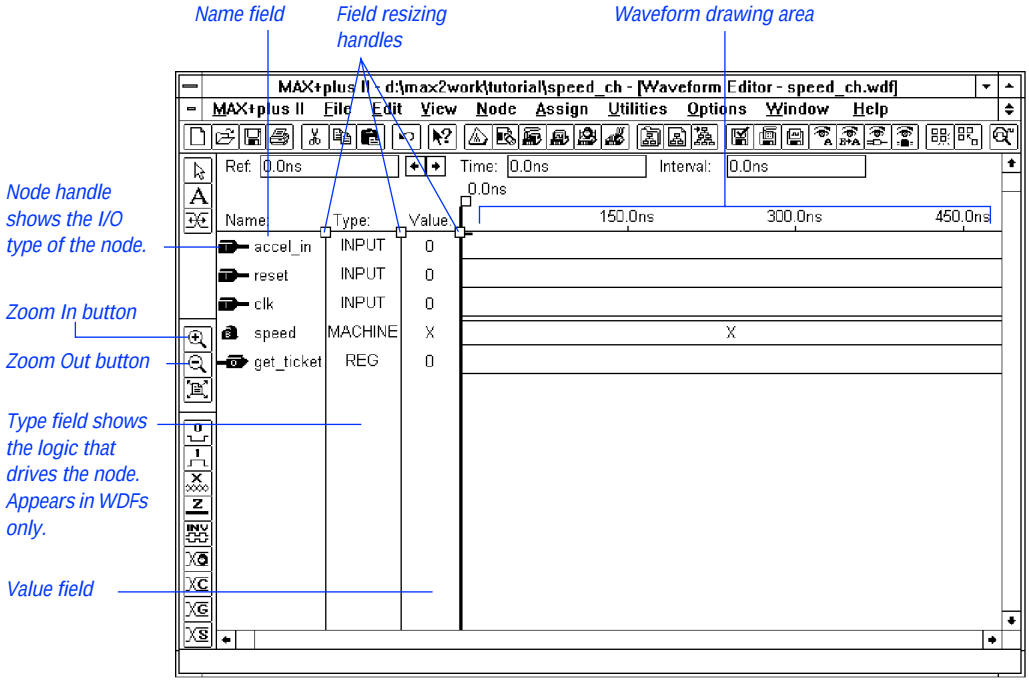
7. Repeat steps 1 through 6 to create the `reset` and `clk` input nodes.

Press F1 when the **Insert Node** dialog box is displayed to get help on the dialog box.

8. Repeat steps 1 through 7 to create the two output nodes `speed` and `get_ticket` with the following characteristics:

Node Name:	Default Value:	I/O Type:	Node Type:	Secondary Inputs:
`speed`	X	Buried Node	Machine	Reset=`reset` Clock=`clk`
`get_ticket`	0	Output Pin	Registered	Clock=`clk`

The new nodes appear as shown in the following illustration:



To display more or less of the waveform drawing area, resize the Name, Type, and/or Value fields by pressing Button 1 on each resizing handle and dragging it to the left or right.

Go to "Buried Nodes," "Input Nodes," and/or "Output Nodes" using **Search for Help on** (Help menu).

3. Set the Grid Size & Show the Grid

To prepare for waveform editing, you will set the grid size and display the grid:

1. Choose **Grid Size** (Options menu). The **Grid Size** dialog box is displayed.

2. Type `30ns` to set the grid at 30 nanosecond intervals. The time unit must immediately follow the time value (i.e., with no space in between).

3. Choose **OK**.

   ☞ In a WDF, the grid size is arbitrary: the time scale indicates only a sequential order of operation, not a specific response time. A specific grid size is used in this tutorial session solely to provide tutorial steps that are easy to follow.

4. If necessary, turn on **Show Grid** (Options menu) to display the dashed vertical grid lines in the window.

   ☞ In a WDF, **Snap to Grid** (Options menu) is always turned on to enable the "magnetic" properties of the grid.

4. Edit the Buried State Machine Node Waveform

To edit the waveform of the buried state machine node `speed`, you will enter three state names on the waveform to represent the transitions between the `legal`, `warning`, and `ticket` states.

☞ As you edit your waveforms, you can change the window display to view larger or smaller portions of the file by clicking Button 1 on the **Zoom In**, **Zoom Out**, and **Fit in Window** buttons on the tool palette, as shown in the illustration on page 198.

To edit the `speed` node:

1. Click Button 1 in the Value field of the `speed` node to select the entire waveform.

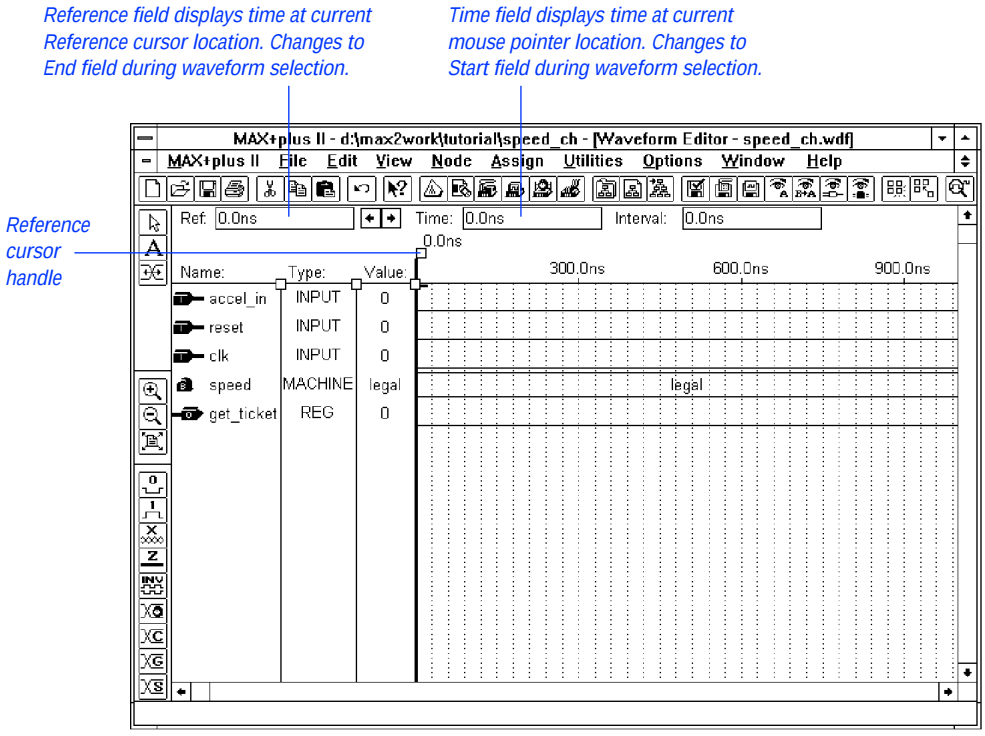
2. Choose the **Overwrite State Name** command (Edit menu) or the **Overwrite State Name** button on the tool palette, as shown in the illustration on page 198.

3. Type `legal` in the *State Name* box.

4. Choose **OK**. The entire waveform is overwritten with the state name `legal`.

5. If necessary, scroll or zoom out to display the grid line at 300 ns.

   See the following illustration:



6. Click Button 1 on the Waveform Editing tool in the tool palette on the left side of the window, as shown in the illustration on page 198, to select it. The Selection pointer changes into the Waveform Editing pointer.

7. Zoom in or out or scroll the window to display the interval from 300 to 540 ns. If you drag to the edge of the window, the file scrolls automatically.

8. Refer to the Time field to find your exact mouse pointer location. With the Waveform Editing pointer, press Button 1 at 300 ns on the `speed` node, and drag the mouse until the interval from 300 to 540 ns is selected. If you drag to the edge of the window, the file scrolls automatically.

   ☞ As you drag the mouse to select a waveform interval, the Time and Reference fields turn into Start and End fields that show the size of the selected interval.

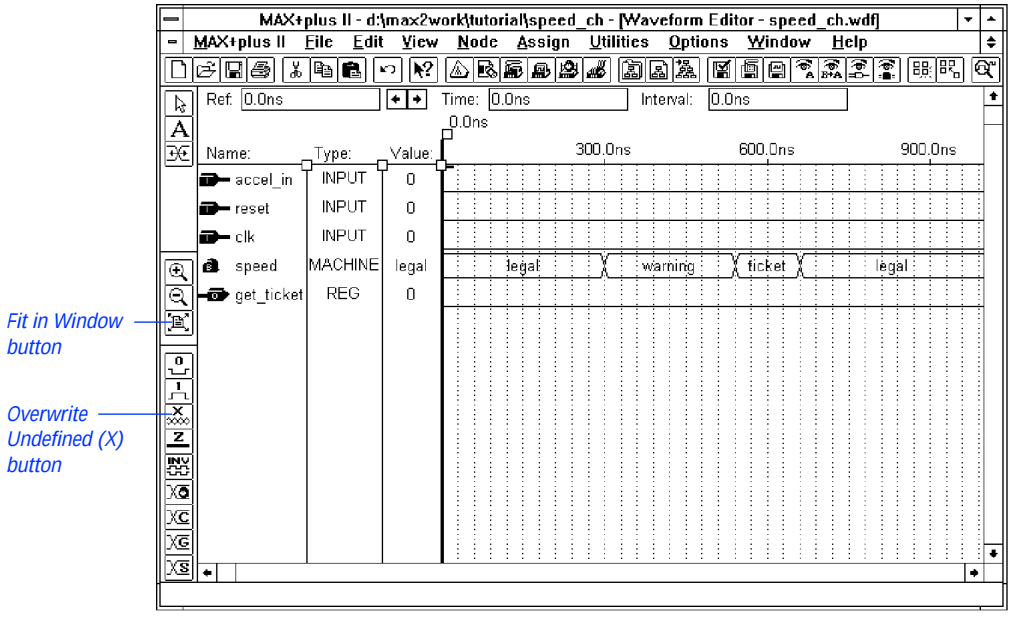
9. Release Button 1. The **Overwrite State Name** dialog box opens automatically because you used the Waveform Editing tool on a state machine waveform.

10. Type `warning` in the *State Name* box.

11. Choose **OK**. The selected interval is overwritten with the state name `warning`.

12. Repeat steps 7 through 11 to overwrite the interval 540 to 660 ns with the state name `ticket`.

The speed node waveform appears as shown in the following illustration (displayed after choosing the **Fit in Window** button from the tool palette to show the full length of each waveform):



5. Edit the Input & Output Node Waveforms

You will now edit the input and output node waveforms. On the `accel_in` waveform, you will overwrite two high logic level intervals (to represent your car's acceleration beyond legal speeds) as the `speed` state machine changes first from `legal` to `warning`, and then from `warning` to `ticket`. You will also overwrite an undefined (X) logic level interval on `accel_in` to mark the transition from the `ticket` state to the `legal` state.

You will overwrite the `clk` signal with a high logic level at alternate 60-ns intervals. You will not change the `reset` signal.

You will edit the `get_ticket` output node by overwriting a high logic level interval on its waveform that corresponds to the `ticket` interval on the `speed` state machine waveform.

To edit the nodes:

1. If necessary, scroll or zoom out to display the 270-ns grid line.

2. With the Waveform Editing tool, press Button 1 at 270 ns on the `accel_in` waveform, drag the mouse until the interval from 270 to 330 ns (i.e., two grid units) is selected, then release Button 1.

   The selected interval is overwritten with a high logic level, which is the complement of the original low logic level. The high logic interval corresponds to the `speed` state machine transition from a `legal` to `warning` state (i.e., if `speed` is in the state `legal` and `accel_in` goes high, then `speed` goes into state `warning`).

   The Waveform Editing tool overwrites an interval on a low or high waveform with its logic level complement. The complement is defined as the opposite of the logic level at the beginning of the interval. High-impedance (Z) and undefined (X) waveforms are also overwritten with low logic levels.

3. Repeat steps 1 and 2 to overwrite a high logic level on the interval from 510 to 570 ns (two grid units) on the `accel_in` waveform, which corresponds to the state machine transition from `warning` to `ticket` (i.e., if `speed` is in the state `warning` and `accel_in` goes high, then `speed` goes into state `ticket`).

4. If necessary, scroll or zoom out to display the 630-ns grid line.

5. Click Button 1 on the Selection tool on the tool palette to activate the Selection pointer. The state machine interval is automatically deselected, and the Reference cursor moves to the beginning of the interval.

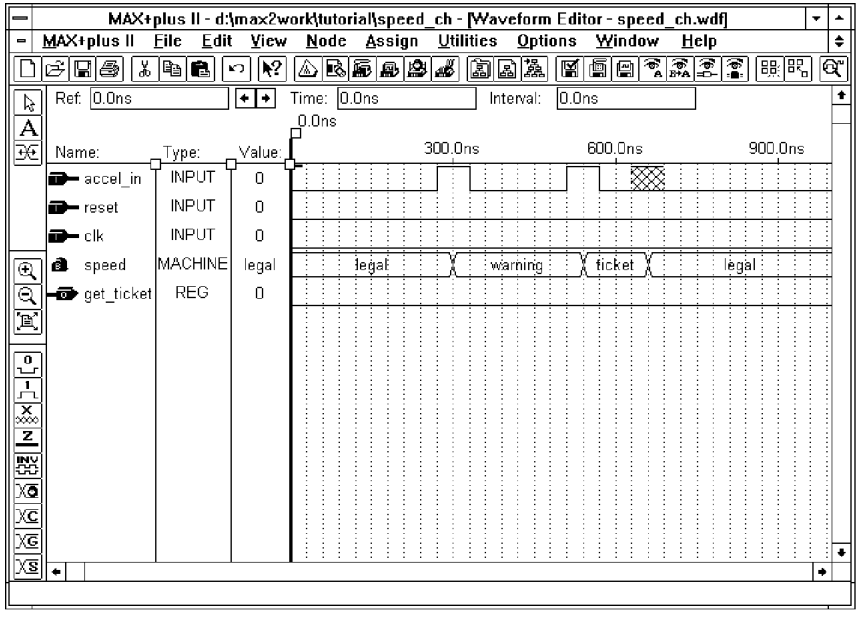
SHORTCUTS

Pressing the Esc key changes the current tool into the Selection tool.

The time at the cursor location is displayed both at the top of the cursor and in the Reference field. The logic levels or state names at this location are shown in the Value field. A dash (-) may appear instead of a value in the Value field if the field is too narrow to display the logic level.

6. With the Selection tool, select the interval from 630 to 690 ns (two grid units). Choose the **Overwrite Undefined (X)** command (Edit menu) or choose the **Overwrite Undefined (X)** button from the tool palette, as shown in the illustration on page 204. The undefined interval corresponds to the state machine transition from `ticket` to `legal` (i.e., `speed` moves from the state `ticket` to `legal` regardless of the value of `accel_in`).

   The `accel_in` node waveform appears as shown in the following illustration:



7. Press Esc to activate the Selection tool again.

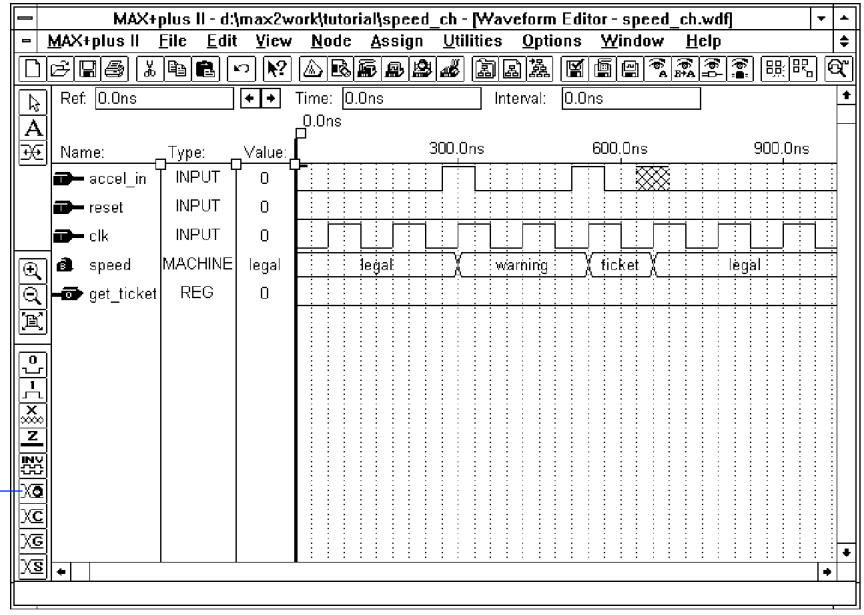
8. Select the entire `clk` node by clicking Button 1 on its node handle or node name, or in the Value field.

9. Choose **Overwrite Clock** (Edit menu). The **Overwrite Clock** dialog box is displayed.

10. Specify 2 in the *Multiplied By* box.

11. Choose **OK** in the **Overwrite Clock** dialog box to accept the default starting value and clock period for `clk`.

The `clk` node waveform appears as shown in the following illustration:



SHORTCUTS

Shortcuts for opening the **Overwrite Clock** dialog box:

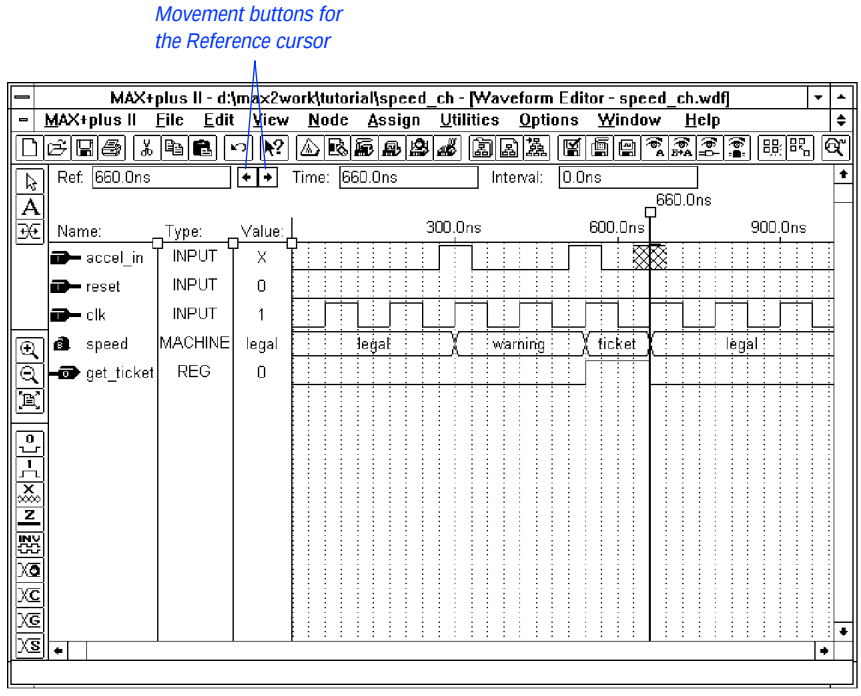
- ✓ Press Button 2 and choose **Overwrite Clock** from the pop-up menu.

or:

- ✓ Choose the **Overwrite Clock** button from the tool palette, as shown in the illustration above.

12. To edit the `get_ticket` node, repeat steps 1 through 3 on page 205 with the Waveform Editing tool to overwrite the interval from 540 to 660 ns on the `get_ticket` waveform with a high logic level. This high logic level corresponds to the `ticket` state on the `speed` node's waveform.

See the following illustration:



6. Confirm the Edits

To confirm your edits, you can move the Reference cursor to each successive logic level transition:

1. If necessary, press Esc to activate the Selection tool.

2. Click Button 1 at 0 ns in the waveform drawing area or drag the Reference cursor by its handle to move the cursor to the beginning of the file.

3. Press the → key or click Button 1 on the right cursor movement button next to the Reference field, as shown in the previous illustration, to move the Reference cursor to the next higher logic level transition. You can also choose **Find Next Transition** (Utilities menu).

4. Repeat as necessary to move the Reference cursor to each successive transition. The logic levels or state names at each transition are displayed in the Value field.

   A dash (-) may appear instead of a value in the Value field if the field is too narrow to display the logic level.

Choose  from the toolbar and click Button 1 on the Reference cursor handle.

7. Check for Basic Errors & Create a Default Symbol

You will now check the file for syntax errors to ensure that it was entered correctly, and then create a default symbol for use in the top-level GDF.

1. Choose **Project Save & Check** (File menu). See "11. Save the File & Check for Basic Errors" on page 183.

2. If **Project Save & Check** is successful, double-click Button 1 on the Document Control Menu box to close the Compiler window.

3. Make sure the **speed_ch.wdf** (or **speed_ch.tdf**) file is displayed in the active window, choose **Create Default Symbol** (File menu), and choose **OK** if you are asked whether it is OK to overwrite the existing Symbol File (**.sym**).

4. Double-click Button 1 on the document icon (or box) to close the file.

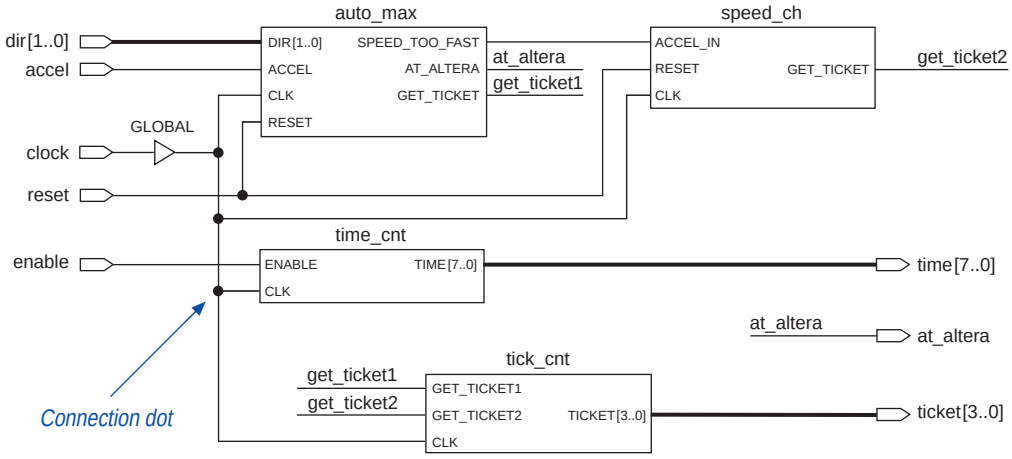
Session 5: Create the Top-Level Graphic Design File

In this session, you will use the MAX+PLUS II Graphic Editor to create the top-level design file for the **chiptrip** project. The **chiptrip.gdf** file incorporates the symbols that represent the four lower-level files, **tick_cnt.gdf**, **auto_max.tdf**, **time_cnt.tdf**, and **speed_ch.wdf** (or **speed_ch.tdf**), created in previous tutorial sessions.

Figure 3-4 shows the **chiptrip.gdf** file, which you will create by following the steps outlined below. This session also introduces additional command shortcuts to help you enter **chiptrip.gdf** quickly. For details about each step, see the procedures described in "Session 2: Create a Graphic Design File."

Some buttons at the right end of the toolbar may be unavailable if your monitor is set to VGA display mode. All toolbar buttons and drop-down lists are available in larger screen displays. If you wish, you can switch to an alternate combination of toolbar buttons for VGA displays. Go to "Setting MAX+PLUS II Preferences" using **Search for Help on** (Help menu) for instructions.

Figure 3-4. chiptrip.gdf



☞ If you skipped the previous tutorial sessions and did not create the four lower-level design files, you can copy the design files and their corresponding Symbol Files from the **\max2work\chiptrip** subdirectory into your **\max2work\tutorial** subdirectory. (On a UNIX workstation, the **max2work** directory is a subdirectory of the **/usr** directory.)

To create **chiptrip.gdf**:

1. Create a new GDF and save it as **chiptrip.gdf** in the **\max2work\tutorial** directory.

2. Specify the project name as **chiptrip**.

SHORTCUTS

Shortcuts for setting the project name:

✓ Choose the **Project Set Project to Current File** button from the toolbar at the top of the window.

or:

✓ Type Ctrl+Shift+J.

3. Enter the symbols for the schematic:

   a. Use the **Enter Symbol** command (Symbol menu) to enter the symbols that represent the lower-level design files created in earlier tutorial sessions:

Symbol Name:	Design File:	Tutorial Session:
`tick_cnt`	**tick_cnt.gdf**	Session 2
`auto_max`	**auto_max.tdf**	Session 3
`time_cnt`	**time_cnt.tdf**	Session 3
`speed_ch`	**speed_ch.wdf**	Session 4

   b. Enter a `GLOBAL` primitive.

   c. Enter five `INPUT` pins and three `OUTPUT` pins.

SHORTCUTS

Shortcuts for opening the **Enter Symbol** dialog box:

✓ With the Selection tool, double-click Button 1 in a blank space.

or:

✓ With any palette tool, press Button 2 in a blank space and choose **Enter Symbol** from the pop-up menu.

4. Name the pins as follows:

Input Pin Names:	**Output Pin Names:**
`dir[1..0]`	`time[7..0]`
`accel`	`at_altera`
`clock`	`ticket[3..0]`
`reset`	
`enable`	

SHORTCUTS

Shortcuts for naming pins:

✓ With the Selection or Text tool, double-click Button 1 on the default pin name and type the desired pin name.

or:

✓ With any palette tool, press Button 2 on the pin symbol, choose **Edit Pin Name** from the pop-up menu, and type the pin name.

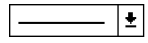
☞ If you press ↵ after you edit a pin name, the next pin name below it is automatically selected for editing.

5. Draw node and bus lines to connect the symbols, as shown in Figure 3-4 on page 210. Be sure to draw all node lines, even those that do not connect symbols together graphically.

SHORTCUTS

Node and bus lines have different line styles. As an alternative to using the **Line Style** command (Options menu), you can select a line style with the **Line Style** drop-down list box on the toolbar or with the Button 2 pop-up menu:

✓ Click Button 1 on the **Line Style** drop-down list box "arrow" and choose the solid line style, which appears at the top of the list:



or:

✓ With any palette tool, press Button 2 on a node or bus line and choose the desired line style from the **Line Style** submenu.

6. To enter connection dots:

   a. With the Selection tool or any drawing tool, click Button 1 on the intersection of two nodes to define an insertion point.

   b. Choose **Toggle Connection Dot** (Edit menu).

SHORTCUTS

Shortcuts for entering (or deleting) a connection dot:

✓ With the Selection tool or any drawing tool, double-click Button 1 on the intersection of two nodes.

or:

✓ With any palette tool, press Button 2 at the intersection of two nodes and choose **Toggle Connection Dot** from the pop-up menu.

or:

✓ With the any palette tool, click Button 1 on the intersection of two nodes and choose the **Toggle Connection Dot** button from the tool palette.

7. Assign names to the unconnected nodes to connect them by name, as shown in the following list and in Figure 3-4.

Symbols are identified here as <*symbol name*>:<*symbol ID*>. Your own symbol ID numbers will vary if you entered the symbols in a different order.

Symbol:	Pinstub/Pin Name:	Node Name:
`auto_max:1`	`AT_ALTERA`	`at_altera`
`auto_max:1`	`GET_TICKET`	`get_ticket1`
`at_altera` (output pin)	`at_altera`	`at_altera`
`speed_ch:2`	`GET_TICKET`	`get_ticket2`
`tick_cnt:3`	`GET_TICKET1`	`get_ticket1`
`tick_cnt:3`	`GET_TICKET2`	`get_ticket2`

Pinstub names are always shown in capital letters in symbols generated with the **Create Default Symbol** command (File menu).

SHORTCUTS

Shortcuts for naming nodes and buses:

✓ With any palette tool, click Button 1 on the node or bus line to define an insertion point and type the desired name.

or:

✓ With any palette tool, press Button 2 on a node or bus line, choose **Edit Node/Bus Name** from the pop-up menu, and type the name.

or:

✓ With the Text tool, type a name on a node or bus line, or type a name in a blank space, then use the Selection pointer to drag the name onto a line to associate the name with the line.

SHORTCUTS

You may wish to customize some or all of your text blocks. As an alternative to using the **Text Size** and **Font** commands (Options menu), you can specify the text size and font with the drop-down list boxes on the toolbar or with the Button 2 pop-up menu.

✓ Click Button 1 on the drop-down list box "arrows" to display the lists of available fonts and sizes, and chose the desired text size or font:

or:

✓ With any palette tool, press Button 2 on a selected node or bus name and choose the desired text size or font from the **Text Size** or **Font** submenus.

8. Choose **Save** (File menu) to save the file.

For a list of additional command shortcuts for the Graphic Editor, go to "Graphic & Symbol Editor Shortcuts" and/or the names of the various commands using **Search for Help on** (Help menu).

Session 6: Compile the Project

In this session, you will compile the **chiptrip** project. The MAX+PLUS II Compiler checks the project for errors, synthesizes the logic, fits the project into an Altera device, generates output files for simulation and programming, and updates the Hierarchy Display window. This session includes the following steps:

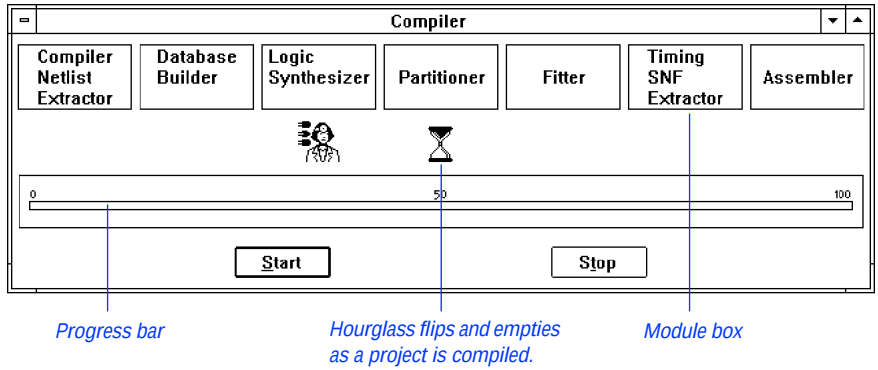
1. Open the Compiler window.
2. Select a device family.
3. Turn on the **Smart Recompile** command.
4. Turn on the Design Doctor utility.
5. Turn on the Security Bit.
6. Select a global project logic synthesis style.
7. Turn on the Timing SNF Extractor.
8. Specify Report File sections to generate.
9. Run the Compiler.
10. Locate the source of a message.
11. Get help on a message.
12. View the Report File.

If you did not complete the earlier tutorial sessions to create the design files for the **chiptrip** project—**tick_cnt.gdf**, **time_cnt.tdf**, **auto_max.tdf**, **speed_ch.wdf**, and **chiptrip.gdf**—you can copy the design files and their corresponding Symbol Files from the **\max2work\chiptrip** subdirectory into the **\max2work\tutorial** subdirectory. (On a UNIX workstation, the **max2work** directory is a subdirectory of thee **/usr** directory.) You must then specify **chiptrip** as the project name. See "2. Specify the Project Name" on page 170.

1. Open the Compiler Window

To open the Compiler window:

✓ Choose **Compiler** (MAX+PLUS II menu). The following illustration shows the Compiler window:



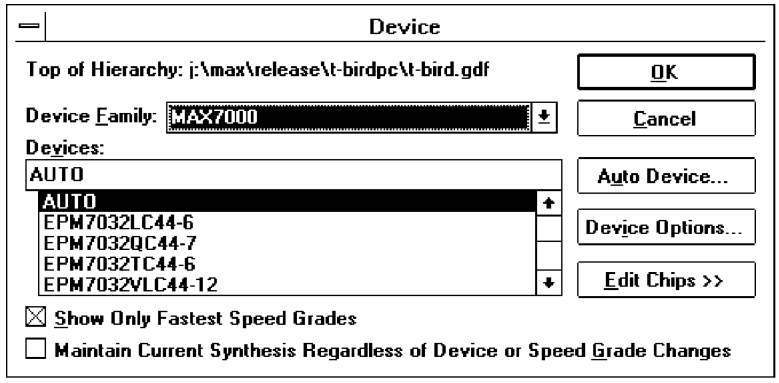
☞ The modules and icons displayed on your screen may differ from those in the illustration, depending on how MAX+PLUS II was set up before you started the tutorial. The Compiler's Processing and Interfaces menus contain commands for turning different modules and utilities on and off.

2. Select a Device Family

You can select any MAX+PLUS II–supported device family for your project. You can also allow the Compiler to automatically choose the most appropriate device within a particular family.

To specify the device family:

1. Choose **Device** (Assign menu). The **Device** dialog box is displayed:



2. If the MAX 7000 family is not already selected, select *MAX7000* in the *Device Family* drop-down list box.

3. If necessary, select *AUTO* in the *Devices* box.

4. Choose **OK**.

3. Turn on the Smart Recompile Command

When the "smart" recompile feature is turned on, the Compiler saves extra database information for the current project for use in subsequent compilations. During "smart" recompilation, the Compiler can determine which modules are not needed to recompile the project, and skip them during recompilation, thereby reducing compilation time.

In "Session 8: View the Fit in the Floorplan Editor" on page 231, you will recompile the **chiptrip** project after changing a pin assignment. Turning on the **Smart Recompile** command now will speed things up when you go through the steps in Session 8.

To turn on the smart recompile feature:

✓ Choose **Smart Recompile** (Processing menu).

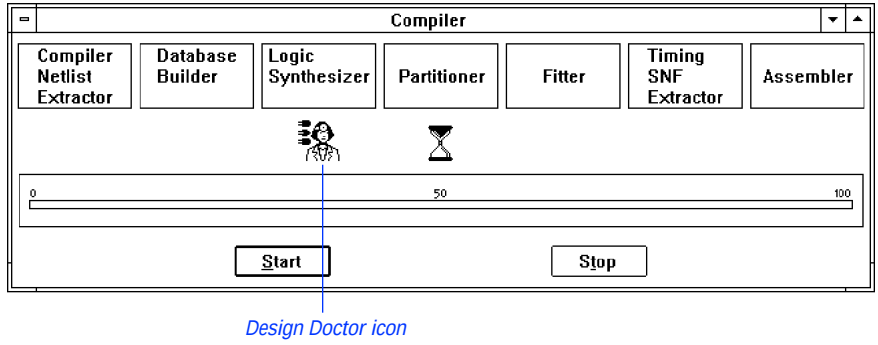
4. Turn on the Design Doctor Utility

During compilation, the optional Design Doctor utility checks all design files in a project for logic that may cause reliability problems in a programmed device.

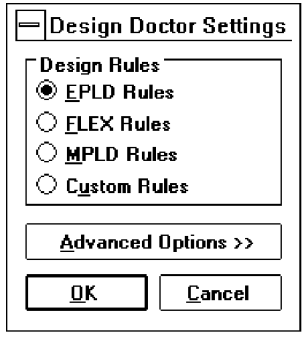
If you have not purchased the Design Doctor utility for MAX+PLUS II, skip to "5. Turn on the Security Bit" on page 220.

To turn on the Design Doctor utility and specify a set of design rules for the analysis:

1. Choose **Design Doctor** (Processing menu). When the command is turned on, a checkmark appears next to the command name on the menu and the Design Doctor icon appears in the Compiler window below the Logic Synthesizer module box, as shown in the following illustration:



2. Choose **Design Doctor Settings** (Processing menu). The **Design Doctor Settings** dialog box is displayed:



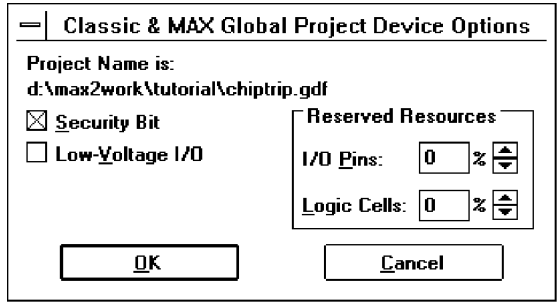
3. If necessary, select *EPLD Rules* and choose **OK**.

Go to "Checking Project Reliability with the Design Doctor" and "Project Reliability Guidelines" using **Search for Help on** (Help menu).

5. Turn on the Security Bit

MAX+PLUS II allows you to specify the default Security Bit setting for all devices in a project. The Security Bit prevents a device from being interrogated.

1. Choose **Global Project Device Options** (Assign menu). The **Classic & MAX Global Project Device Options** dialog box is displayed:



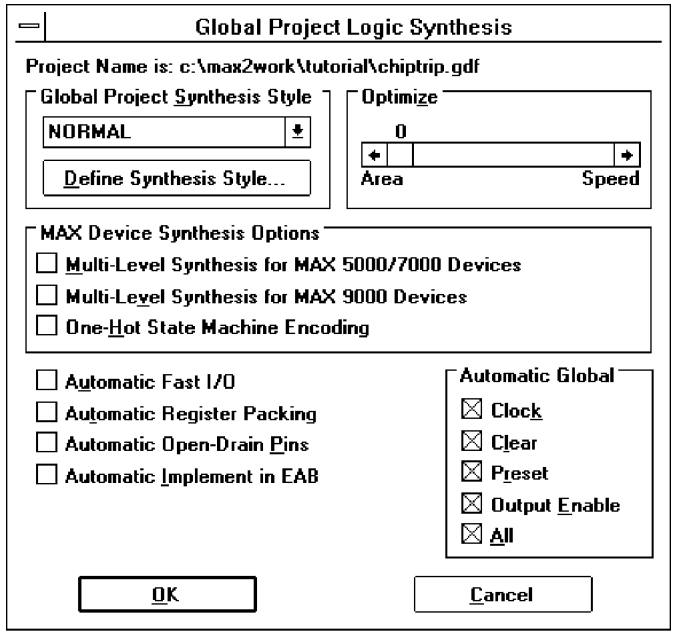
2. If necessary, turn on *Security Bit* and choose **OK**.

6. Select a Global Project Logic Synthesis Style

You can select a logic synthesis style for the project that guides the Compiler's Logic Synthesizer module during compilation. The default logic synthesis style for a new project is "Normal." The logic option settings in this style optimize your project logic for minimum silicon resource usage.

To select a logic synthesis style for the project:

1. Choose **Global Project Logic Synthesis** (Assign menu). The **Global Project Logic Synthesis** dialog box is displayed:



2. If necessary, select *Normal* in the *Global Project Synthesis Style* drop-down list box and choose **OK**.

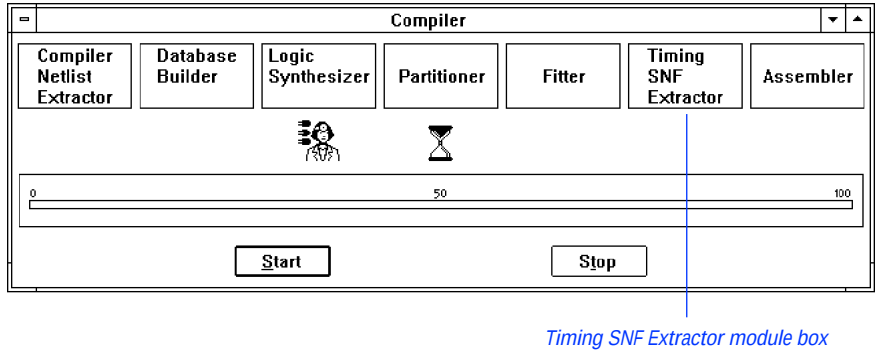
Go to "Specifying Global Project Logic Synthesis Settings" using **Search for Help on** (Help menu).

7. Turn on the Timing SNF Extractor

The Compiler can create a Simulator Netlist File (**.snf**) that contains the logic and timing information used by the MAX+PLUS II Simulator and Timing Analyzer. A timing SNF is a binary file that contains all logic and timing information required for simulation, delay prediction, and timing analysis.

To turn on the Timing SNF Extractor module:

✓ Choose **Timing SNF Extractor** (Processing menu). When the command is turned on, a checkmark appears next to the command name on the menu and the Timing SNF Extractor module box appears in the Compiler window, as shown in the following illustration:



Timing SNF Extractor module box

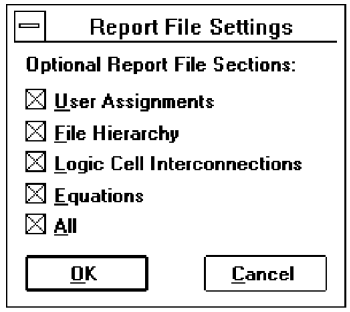
☞ If the module boxes for the EDIF, VHDL, and/or Verilog Netlist Writer are displayed, turn these modules off with the **EDIF Netlist Writer**, **VHDL Netlist Writer**, and/or **Verilog Netlist Writer** commands (Interfaces menu).

8. Specify Report File Sections to Generate

The Report File (**.rpt**), which is generated by the Compiler's Fitter module, shows how device resources are used in the **chiptrip** project. The Compiler allows you to specify which optional information should be included in a Report File.

To specify that all sections should be included in the Report File:

1. Choose **Report File Settings** (Processing menu). The **Report File Settings** dialog box is displayed:



2. If necessary, turn *All* on and choose **OK**.

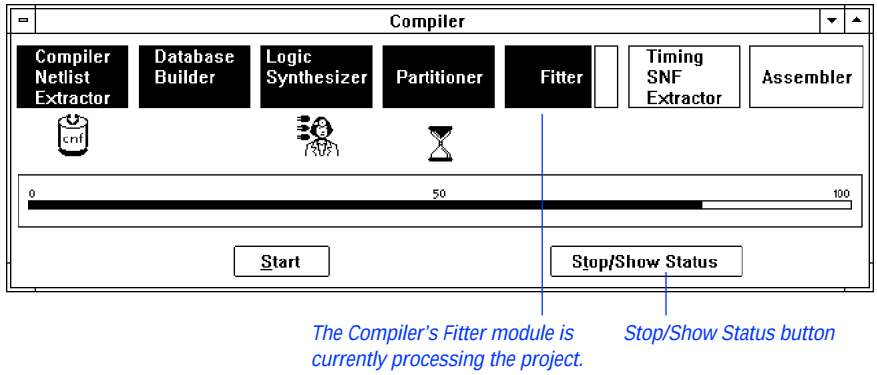
Go to "Report File Format" using **Search for Help on** (Help menu).

9. Run the Compiler

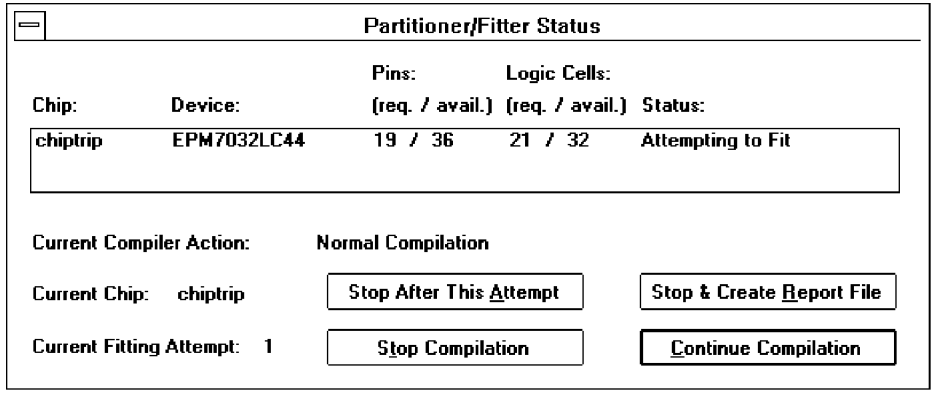
To compile the project:

1. Choose the **Start** button. As the Compiler processes the **chiptrip** project, any information, error, or warning messages appear in a Message Processor window that opens automatically.

The **Stop/Show Status** button appears in place of the **Stop** button when the Compiler's Partitioner and Fitter modules are processing the project, as shown in the following illustration:



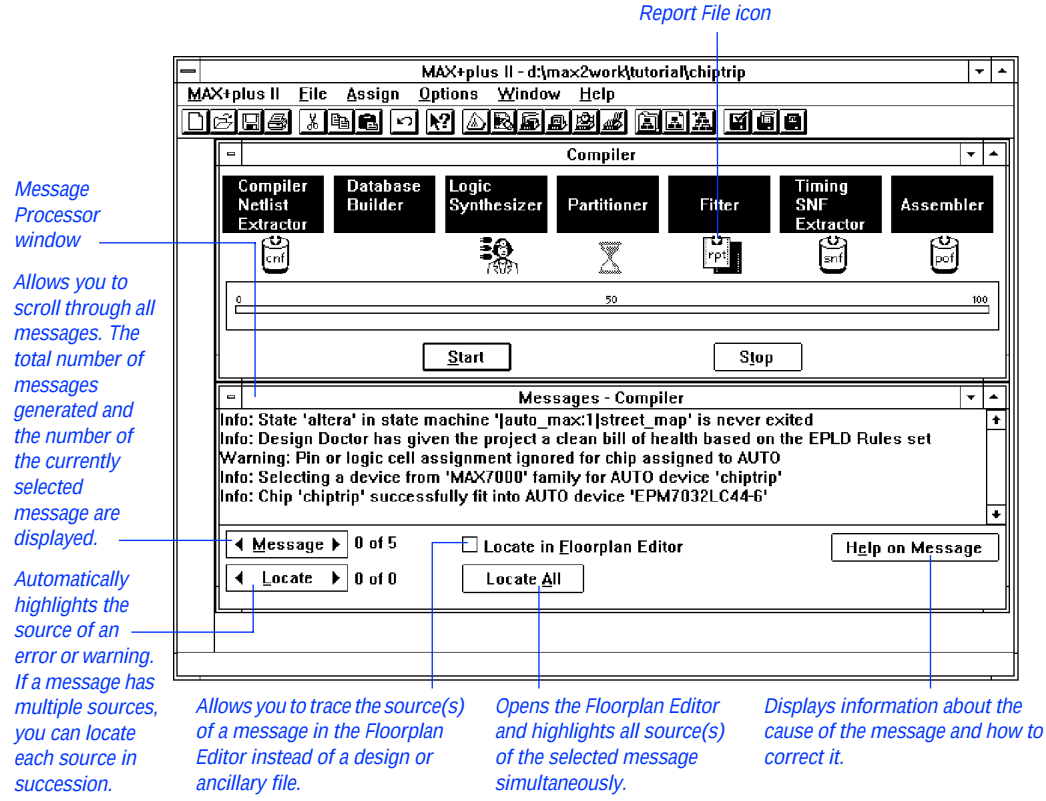
2. While the Fitter module is processing the project, choose the **Stop/Show Status** button in the Compiler window. The **Partitioner/Fitter Status** dialog box is displayed, listing all chips in the project and their fitting status:



3. Choose the **Continue Compilation** button in the **Partitioner/Fitter Status** dialog box to resume compilation.

   The Compiler runs in the background, freeing your computer for other work. However, this compilation is short and you will not have to wait long for it to finish. When you compile a larger, more complex project, you can start a compilation and then switch to another application to continue your work.

When compilation is finished, icons representing the output files generated by the Compiler appear below the module boxes. You can open any of these output files by double-clicking Button 1 on the appropriate file icon. The **chiptrip** compilation processing also generates four information messages, as shown in the following illustration:



As shown in the illustration, the Design Doctor has given the **chiptrip** project a "clean bill of health," and the Compiler has selected the EPM7032LC44-6 device (an EPM7032-6 device in a 44-pin plastic J-lead chip carrier package) for the project.

10. Locate the Source of a Message

You can direct the Message Processor to locate the source of a message within the relevant design file.

To locate the source of a message:

1. If necessary, switch to the Message Processor window by choosing **Message Processor** (MAX+PLUS II menu).

2. Click Button 1 on the first message or on the right side of the **Message** button to select the first message: `Info: State 'altera' in the state machine '|auto_max:1|street_map' is never exited`.

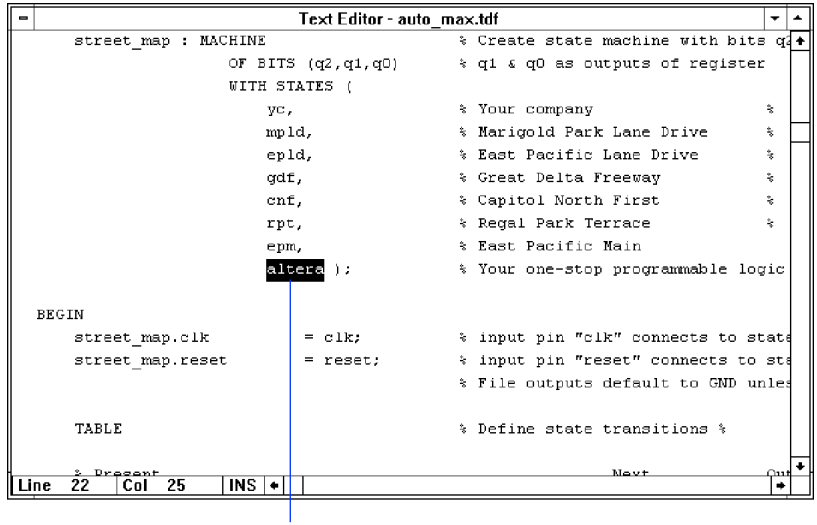
3. If necessary, turn off the *Locate in Floorplan Editor* option.

4. Choose the **Locate** button.

SHORTCUTS

Double-clicking Button 1 on a message is a shortcut for choosing the **Locate** button.

The Message Processor automatically opens the TDF containing the source of the message, and highlights the source's location within the design file, as shown in the following illustration:



Source of the message is highlighted in the original design file.

5. If the message has multiple sources, you can locate the additional sources by choosing **Locate** again.

6. Once you finish viewing the design file(s), close the design editor window(s) to return to the Message Processor window.

11. Get Help on a Message

You can get instant context-sensitive help on the cause of a message.

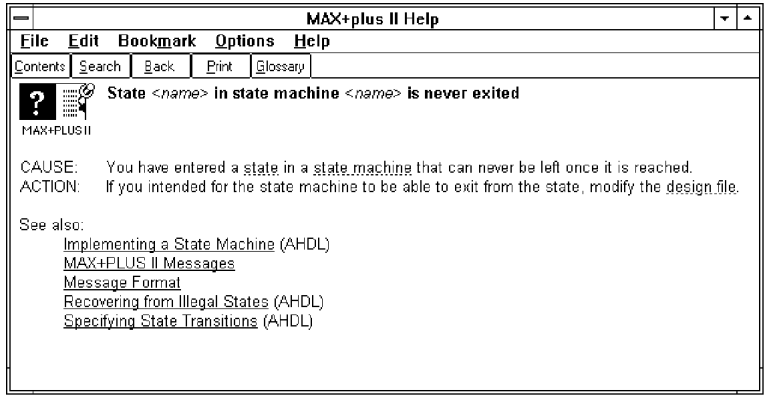
To get help on a message:

1. Select a message.

2. Choose the **Help on Message** button in the Message Processor window.

   The Message Processor opens the MAX+PLUS II Help window, which shows the cause of the message and any action you must take to correct the problem.

   The following illustration shows the Help topic for the first message in the Message Processor window:



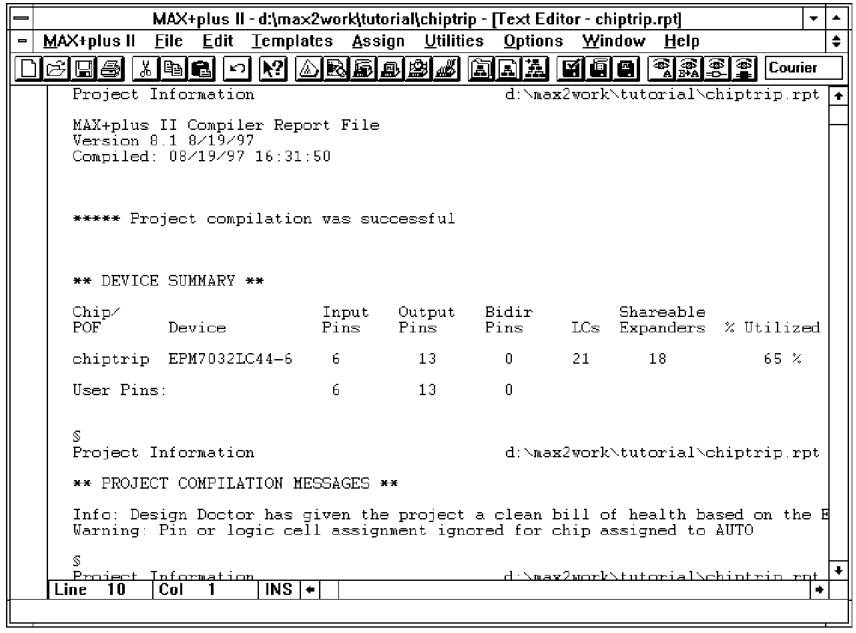
3. Once you finish viewing the Help topic, close the Help window and return to the Compiler window.

12. View the Report File

The Report File (**.rpt**) contains two types of information about the **chiptrip** project: project-wide information (sections entitled Device Summary, Project Compilation Messages, File Hierarchy, etc.) and device-specific information (sections entitled Resource Usage, Routing Resources, Logic Cell Interconnections, etc.). You can open the Report File for the current compilation directly from the Compiler window.

To open the Report File:

1. Double-click Button 1 on the Report File icon in the Compiler window, as shown in the illustration on page 225. The Report File is displayed in a Text Editor window, as shown in the following illustration:



Choose on the toolbar and click on a section heading delimited by ** characters to go to context-sensitive help on specific sections of the Report File.

2. Once you finish viewing the Report File, close it to return to the Compiler window.

3. Close the Compiler window.

Session 7: View the Project in the Hierarchy Display

In this session, you will view the hierarchy of the **chiptrip** project in the Hierarchy Display window. This session includes the following steps:

1. Open the Hierarchy Display window.
2. Bring **chiptrip.gdf** to the front.
3. Close any open file(s).

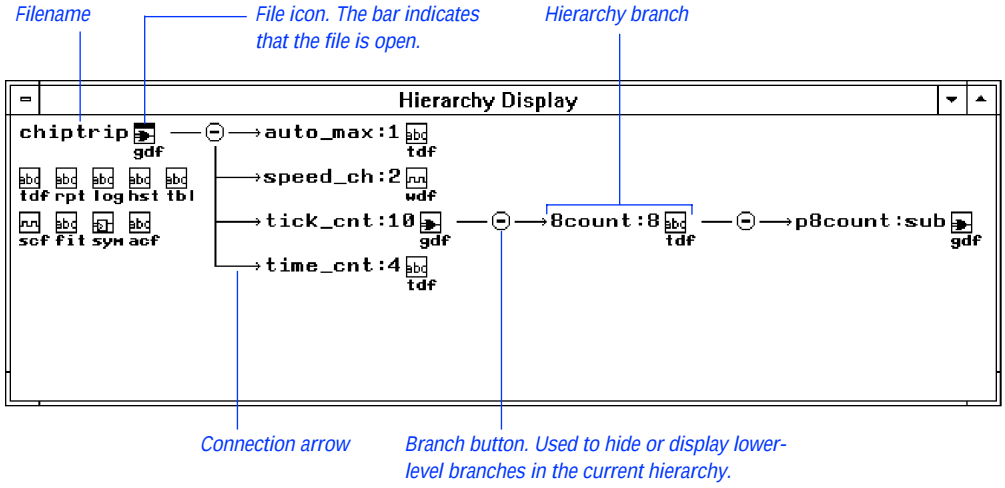
1. Open the Hierarchy Display Window

To see the **chiptrip** hierarchy:

✓ Choose **Hierarchy Display** from the MAX+PLUS II menu.

Each file in the project hierarchy tree is represented in the Hierarchy Display by its filename and a file icon that indicates the file type. A bar at the top of a file icon indicates an open file.

See the following illustration:



Choose from the toolbar and click Button 1 on any item in the window to go to MAX+PLUS II Hierarchy Display Help on that item.

2. Bring chiptrip.gdf to the Front

The Hierarchy Display window allows you to quickly open or bring to the front any design file in the project hierarchy, or any ancillary file with the same filename as the project. When you open a file from the Hierarchy Display, MAX+PLUS II automatically opens the appropriate editor.

To bring **chiptrip.gdf** to the front:

✓ Double-click Button 1 on the GDF icon next to the **chiptrip** filename. The Graphic Editor window with **chiptrip.gdf** comes to the front.

Go to "Navigating the Hierarchy" using **Search for Help on** (Help menu).

3. Close any Open File(s)

To close any open files in the Hierarchy Display:

1. Bring the Hierarchy Display window back to the front. The **chiptrip.gdf** icon has a bar over it, indicating that the file is open.
2. Click Button 1 on the **chiptrip.gdf** icon to select it.

    You can select multiple files by pressing the Shift key while clicking Button 1 on file icons.

3. Choose **Close Editor** (File menu). The **chiptrip.gdf** file is closed and the bar above its icon disappears.
4. Double-click Button 1 on the document icon (or box) to close the Hierarchy Display window.

Go to "Selecting a File Icon" using **Search for Help on** (Help menu).

Session 8: View the Fit in the Floorplan Editor

In this session, you will use the Floorplan Editor window to view Compiler partitioning and fitting results, as well as to enter and edit physical device resource assignments for your project. You will also view the Compiler's logic placement, compare your own assignments to the Compiler's assignments, and back-annotate the results of compilation. This session includes the following steps:

1. Open the Floorplan Editor window.
2. Back-annotate the project and edit assignments.
3. Recompile the project.
4. Display routing information in the Floorplan Editor window.
5. Display equation and routing information with the Report File Equation Viewer.

Go to "Floorplan Editor Procedures" using **Search for Help on** (Help menu) for more information on using the Floorplan Editor.

1. Open the Floorplan Editor Window

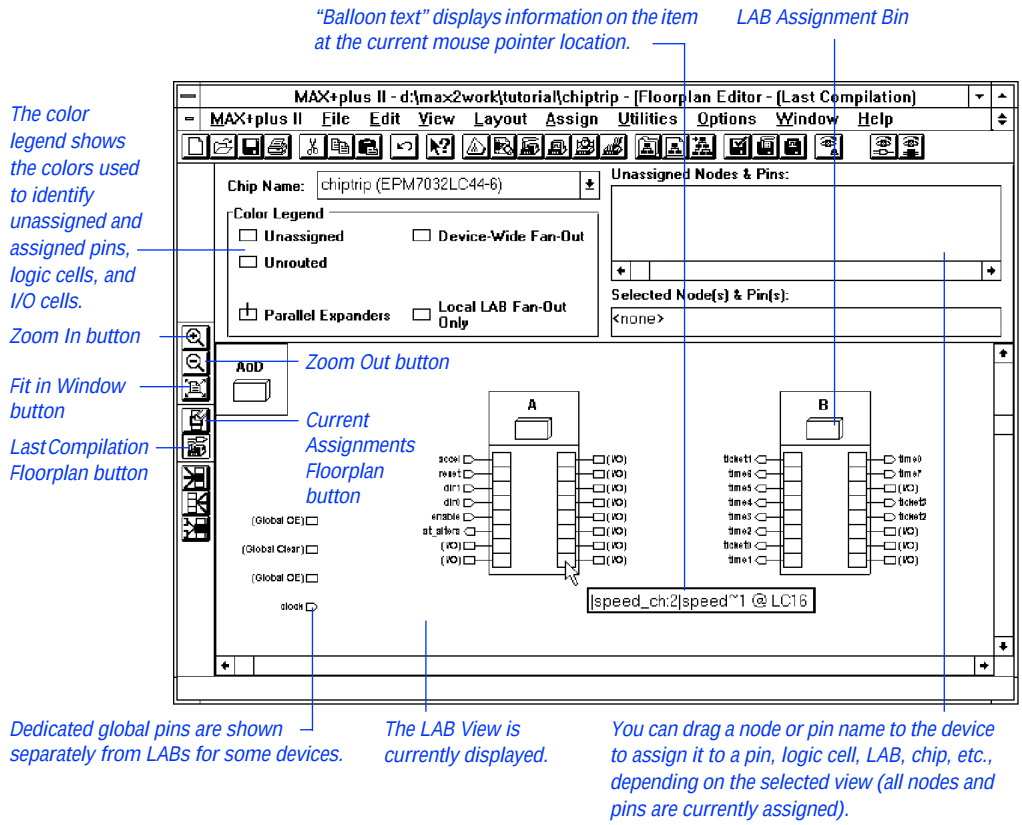
The Floorplan Editor provides two displays—the Device View and the LAB View. The Device View shows all pins on a device package and their function. The LAB View shows the interior of the device, including all LABs; the individual logic cells within each LAB; and I/O cells, embedded cells, and Embedded Array Blocks (EABs) if they are available in the target device. (The EPM7032 device used in this project does not include I/O cells, embedded cells, or EABs.) The LAB View also displays pin locations so that you can see the relationships between pins and logic resources in the interior of the device.

To view **chiptrip** in the Floorplan Editor window:

1. Choose **Floorplan Editor** from the MAX+PLUS II menu. The Floorplan Editor opens and displays the view that was last used to examine the floorplan of the device selected for the project.

2. If necessary, click Button 1 on the **Maximize** button in the Floorplan Editor title bar to maximize the window.

3. If necessary, choose the **LAB View** and **Last Compilation Floorplan** commands from the Layout menu. The **chiptrip** project is displayed in the Floorplan Editor window, as shown in the following illustration:



SHORTCUTS

Double-clicking Button 1 in a blank space in the Floorplan Editor window switches back and forth between the LAB and Device View displays.

Clicking Button 1 on the **Last Compilation Floorplan** button on the tool palette is a shortcut for choosing the **Last Compilation Floorplan** command (Layout menu).

2. Back-Annotate the Project & Edit Assignments

The Floorplan Editor allows you to view and edit your current assignments, which are stored in the project's Assignment & Configuration File (**.acf**). After you have compiled the project, you can edit the Compiler's assignments, which are stored in the project's Fit File (**.fit**), by back-annotating the project and then changing the current floorplan assignments.

To back-annotate your project:

1. Choose **Back-Annotate Project** (Assign menu). The **Back-Annotate Project** dialog box is displayed.

2. Turn on the *Chips, Logic Cells, Pins & Devices* option under *Back-Annotate to ACF* to back-annotate all assignments.

3. Choose **OK**. MAX+PLUS II copies the pin, logic cell, chip, and device assignments from the Fit File into the ACF, overwriting the previous assignments.

4. Choose **Current Assignments Floorplan** (Layout menu). The Floorplan Editor window displays the current assignments for the **chiptrip** project.

If some logic is unassigned in your project, the Floorplan Editor provides a list of unassigned node and pin names, as shown in the previous illustration. Each name has a "handle" that you can drag to an individual pin or logic cell—or to a more general assignment "bin"—in the Device View or LAB View display. You can also drag a node or pin with an existing assignment back to the list of unassigned nodes or to a different location on the device.

You can easily edit your current assignments in the Floorplan Editor window. In this example, you will reassign the `clock` pin to a new location and recompile the project.

To edit the `clock` pin assignment:

1. Choose **Find Text** (Utilities menu). The **Find Text** dialog box is displayed.

2. Type `clock` in the *Search For* box.

3. Turn off the *All* option under *Types of Text to Find*.

4. Turn on the *Pin & Node Names* option under *Types of Text to Find*.

5. Choose **OK**. The `clock` pin assignment is highlighted and the information "`clock@43(Global CLK)`" is displayed in the *Selected Node(s) & Pin(s)* field in the Floorplan Editor window.

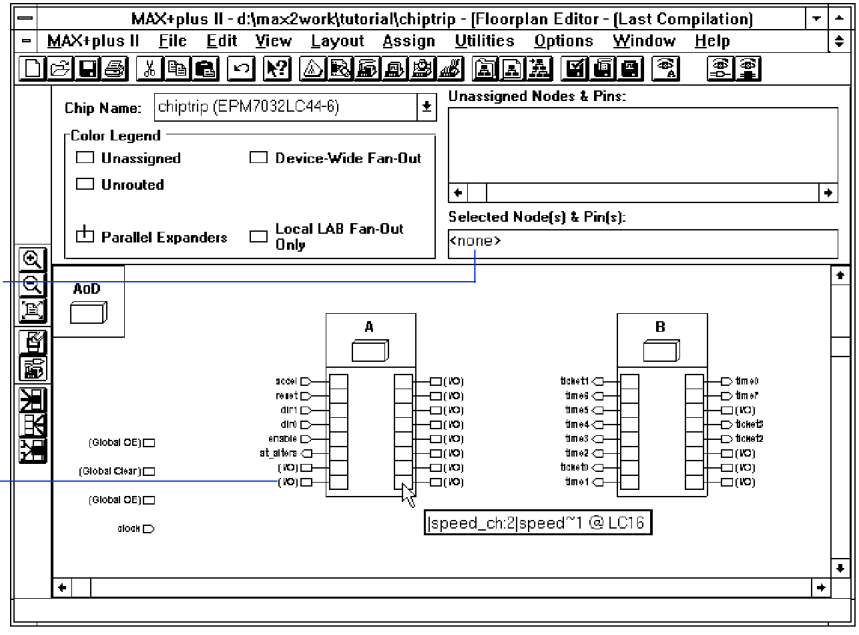
On the EPM7032LC44 device, pin 43 is the dedicated global Clock pin. The Compiler automatically assigned the `clock` signal to this pin when the project was compiled in Session 7.

6. Turn on **Show Moved Nodes in Gray** (Options menu).

7. With Button 1, drag the selected `clock` pin assignment from pin 43 to an unassigned I/O pin of your choice. The assignment is shown in gray at its new location, as shown in the following illustration:

The assignment and location of a selected item are displayed in the Selected Node(s) & Pin(s) field.

Reassigned Clock pin in a new location.



Go to "Back-Annotating Assignments for a Project" and "Finding Nodes & Pins in the Floorplan Editor" using **Search for Help on** (Help menu) for more information on using the Floorplan Editor.

3. Recompile the Project

Once you have edited the pin assignment for the `clock` input pin, you must recompile **chiptrip** to verify whether or not your new assignment is legal for the EPM7032LC44 device.

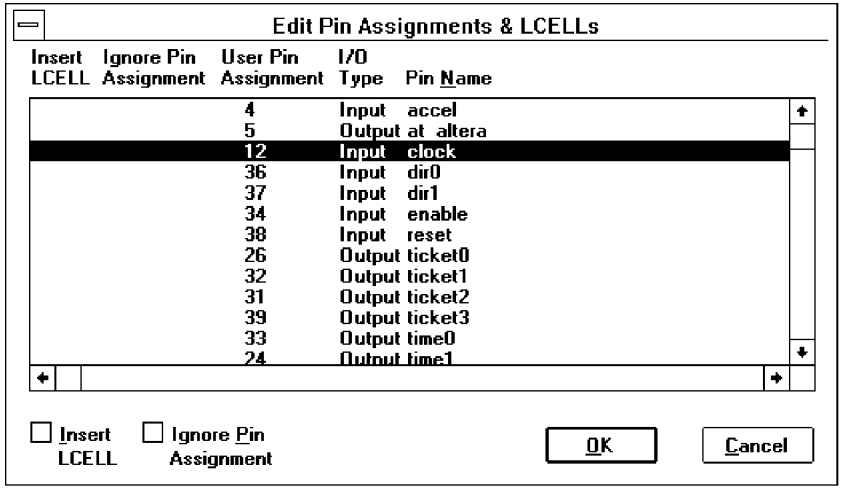
1. Open the Compiler window by choosing **Compiler** (MAX+PLUS II menu).

2. Choose **Start**. The Compiler begins processing the project, based on the new pin assignment. The Compiler then halts, informing you that the project doesn't fit, and asks if you wish to override some existing settings and/or assignments.

3. Choose **Yes**. The **Override User Assignments** dialog box appears, displaying the following message: `Illegal assignment -- 'clock' on pin` *<number>*.

4. Choose the **Edit Pin Assignments & LCELLs** button. The **Edit Pin Assignments & LCELLs** dialog box is displayed:

   a. Click Button 1 on the `clock` assignment in the list box to select it, as shown in the following illustration:



   b. Turn on the *Ignore Pin Assignment* option at the bottom of the dialog box.

c. Choose **OK** to close the **Edit Pin Assignments & LCELLs** dialog box.

5. Choose **OK** to close the **Override User Assignments** dialog box. The Compiler continues processing the project and, when it finishes, displays a message indicating that project compilation was successful.

6. Choose **Floorplan Editor** (MAX+PLUS II menu) to return to the Floorplan Editor window.

7. Choose **Last Compilation Floorplan** (Layout menu). The `clock` pin assignment now reappears on pin 43.

8. Back-annotate the Compiler's assignments, as described in steps 2 through 4 on page 234.

4. Display Routing Information in the Floorplan Editor Window

The Floorplan Editor allows you to view the routing information for one or more selected logic cells, pins, and assignment bins using a variety of different methods. You can also view routing statistics for any part of the current chip.

To display node fan-in and fan-out routing information:

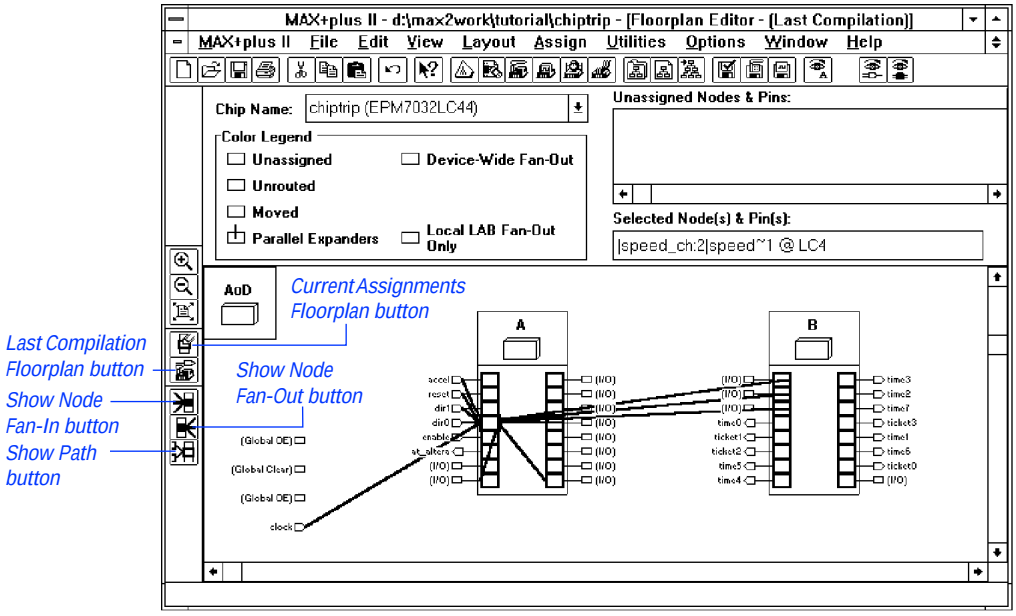
1. Turn on **Show Node Fan-In** and/or **Show Node Fan-Out** (Options menu).

SHORTCUTS

Choose the **Show Node Fan-In** and **Show Node Fan-Out** buttons on the tool palette, as shown in the next illustration.

2. Go to the LAB View and select one or more logic cells, pins, or assignment bins.

The Floorplan Editor displays the fan-in (pink) and/or fan-out (blue) routing lines that apply to the selected item(s). The following illustration shows the fan-in and fan-out of the selected `|speed_ch:2|speed~1` node at LC4 (logic cell 4):



Fan-in and fan-out lines are updated automatically when you move an assignment. In addition, if you move a node to a new location, the assignment color changes if the **Show Moved Nodes in Gray** command (Options menu) is turned on.

You can also choose to view only the signal paths between two or more items, without additional fan-in and fan-out information:

✓ Turn on **Show Path** (Options menu).

This command allows you to view only the connections between the selected nodes, and is especially useful for tracing critical timing paths. When **Show Path** is turned on, **Show Node Fan-In** and **Show Node Fan-Out** are turned off automatically, and vice versa.

SHORTCUTS

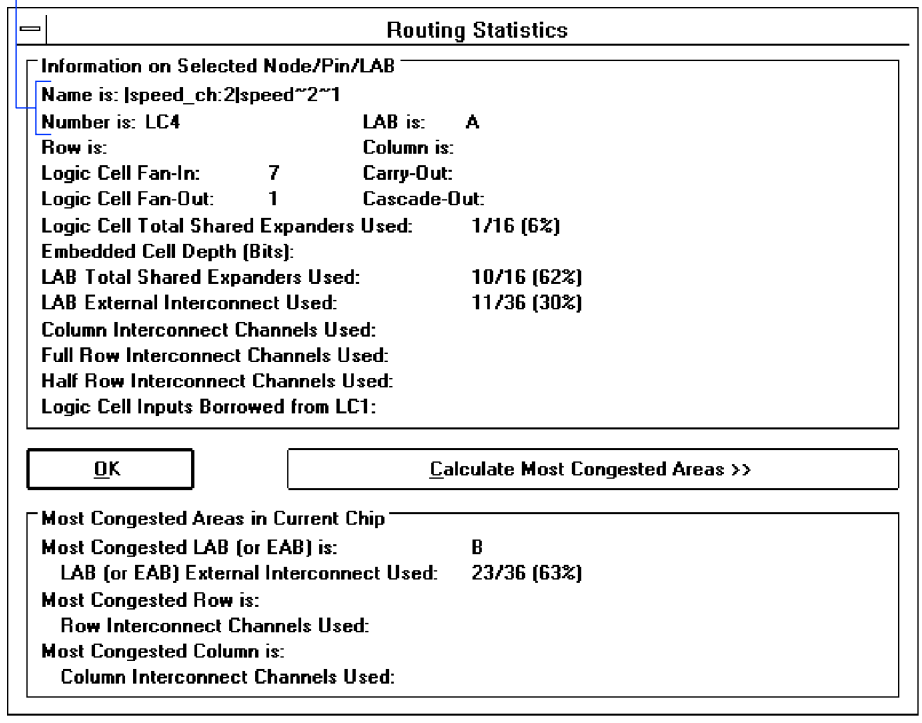
Choose the **Show Path** button on the tool palette, as shown in the previous illustration.

To display detailed routing statistics for one or more logic cells, pins, or assignment bins:

1. Select one or more logic cells, pins, or assignment bins.

2. Choose **Routing Statistics** (Options menu), then choose the **Calculate Most Congested Areas** button. The **Routing Statistics** dialog box is displayed, as shown in the following illustration:

Routing Statistics for the |speed_ch:2|speed~1 node are shown in this example.



SHORTCUTS

Double-clicking Button 1 on a single item is a shortcut for opening the **Routing Statistics** dialog box.

3. Choose **OK** to close the **Routing Statistics** dialog box.

5. Display Equation & Routing Information with the Report File Equation Viewer

The Floorplan Editor includes a Report File Equation Viewer that allows you to view the Report File (**.rpt**) equations, and fan-in and fan-out information for pin and logic cell assignments. You can view this information in two ways: by selecting individual assignments in the Floorplan Editor window; and by jumping to associated assignments within the Report File Equation Viewer window. This equation viewer allows you to examine the logic that feeds or is fed by any node in the project. As an example, you will view information for the `time2` pin and then jump to the `time0` pin using the Report File Equation Viewer.

To view the equation(s) for the `time2` pin:

1. Choose **Full Screen** (Layout menu) to allow more room to display the chip.

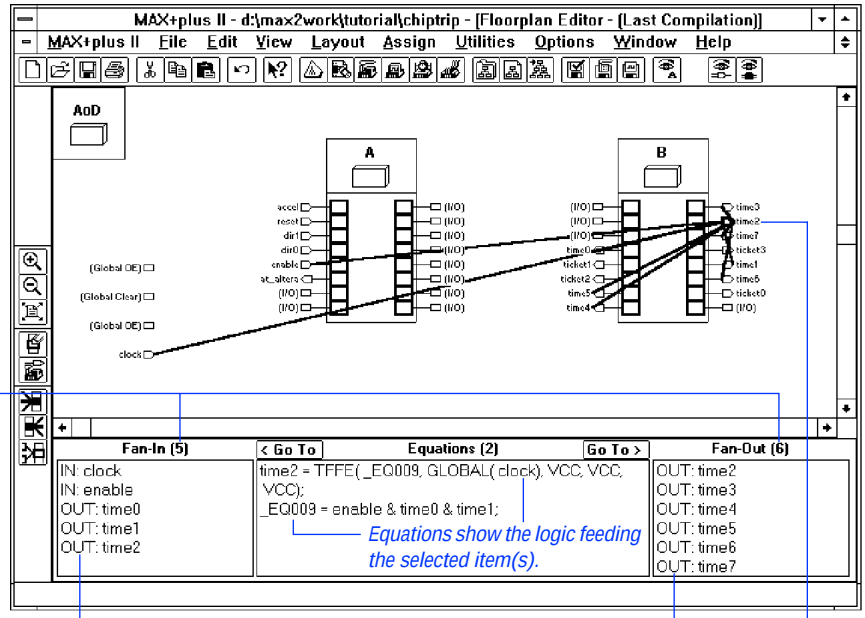
2. Choose **Report File Equation Viewer** (Layout menu). The Report File Equation Viewer window appears at the bottom of the Floorplan Editor window.

3. Click Button 1 on the `time2` pin in the Floorplan Editor window. The text `time2@31(I/O)` appears in balloon text when the mouse pointer is over the correct pin (pin 31).

   Pin numbers may vary when you create the **chiptrip** project.

   The equation and routing information for `time2` appears in the Report File Equation Viewer window, as shown in the following illustration:



If the **Show Node Fan-In** and **Show Node Fan-Out** commands (Options menu) are turned on, the Floorplan Editor displays the fan-in and fan-out lines that correspond to the items listed in the Report File Equation Viewer window.

4. While equation and routing information for the `time2` pin is displayed in the Report File Equation Viewer window, select the "`OUT: time0`" pin under *Fan-In* with Button 1 and choose the **< Go To** button. The equation and routing information for the `time0` pin then appears in the Report File Equation Viewer window. In addition, the `time0` pin is highlighted in the Floorplan Editor window, and its fan-in and fan-out lines are displayed.

SHORTCUTS

Double-clicking Button 1 on a node in the *Fan-In* or *Fan-Out* section of the Report File Equation Viewer window is a shortcut for selecting the node and choosing the **Go To** button.

Go to "Floorplan Editor Procedures" using **Search for Help on** (Help menu) for more information on working with the Floorplan Editor.

Simulation Overview

Your logic circuit has compiled without errors. However, only a simulation will confirm that it behaves exactly as you desire. Skipping simulation is like buying a car without taking it for a test drive: you may be reasonably sure that it works, but does it actually fit your particular needs?

If you have not purchased the simulation tools for MAX+PLUS II, please proceed to "Session 12: Analyze Timing" on page 266.

What is Simulation?

Design entry and compilation are only part of the design process. Simulation is equally important, if not more so. Successful compilation only guarantees that a programming file will be created for your project, not that the project will perform as you expect. Yet, many designers avoid simulation because they think it is too slow, or too difficult to learn. Learning to simulate with MAX+PLUS II, however, is as easy as learning design entry and compilation. Since simulation provides the quickest, easiest way to verify your project's performance, you can save both time and effort.

You simulate a project to verify that it functions correctly. Simulation allows you to thoroughly test your project to ensure that it responds correctly in every possible situation before you program it into a device.

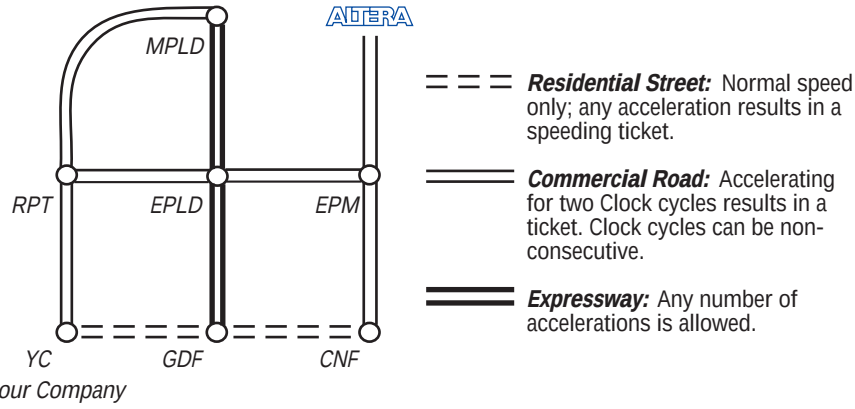
During simulation, you supply input vectors to the MAX+PLUS II Simulator. The Simulator uses these inputs to create the output signals that a programmed device would produce under the same conditions. In a typical simulation session, you create multiple sets of input vectors to check the resulting outputs.

Depending on the kind of information you need, you can perform functional, timing, or linked multi-project simulation with MAX+PLUS II. Functional simulation tests only the logical operation of a project, while timing simulation tests both the logical operation and the worst-case timing of the target device(s). Linked simulation combines the functional and timing information from multiple projects to allow you to perform a board-level-type simulation. In this tutorial, you will perform a timing simulation.

How Does the Chiptrip Simulation Work?

The **chiptrip** tutorial simulation sessions are set up as a driving simulation game. The inputs to **chiptrip** (e.g., direction, acceleration, Clock cycles) produce simulation outputs that correspond to intersections on the map in Figure 3-5. Thus, by creating the right inputs, you can determine the path you take as well as your speed.

Figure 3-5. Map to Altera



As in real life, there are rules for handling your car (the laws of physics) and for driving on each type of road (the laws of the state). Needless to say, there are also plenty of consequences if you break the rules.

You & Your Vehicle

Direction and acceleration are your main concerns. The directional inputs control the next location of your vehicle, based on its present location. For example, if you start at `YC` (Your Company) and travel one block east and one block north, you would drive past `GDF` to reach `EPLD`.

The acceleration input moves your car at one of two speeds in the direction you have chosen. When `accel` is low, you travel at normal speed, one block per Clock cycle (e.g., from `YC` to `GDF`). When `accel` is high, you accelerate through two blocks in a single Clock cycle (e.g., from `YC` to `CNF`).

As you drive, the `clock` input ticks off the time units, and the `enable` input allows your time counter to count. By keeping `enable` high, you can time yourself and see how long it takes to get across town.

The Roads

You can travel along three types of roads, as shown in Figure 3-5. Residential roads are narrow, with houses on both sides, and children playing in the street. Don't be lulled by the peaceful suburban vista, however; these roads are notorious speed-traps. If you accelerate just once, the police will give you a ticket.

Commercial streets, such as the one leading to Altera via `CNF` and `EPM`, are in typical downtown business areas. You can speed once and get a warning. If you speed a second time, however, you *will* get a ticket.

The expressway has five spacious lanes in each direction. Traffic speeds along and backups are rare. (This condition does not reflect reality: if you do visit Altera and find yourself on U.S. Highway 101, find a pleasant radio station.) Accelerate at will. Don't worry—the police will not stop you.

Simulation Goals

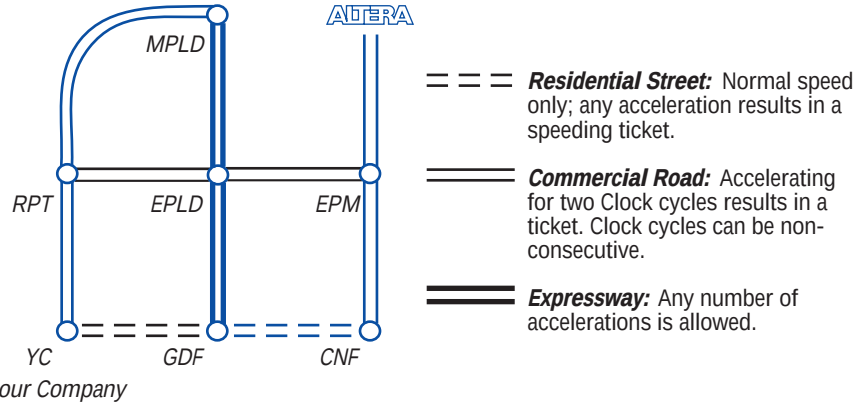
You will set up your simulation inputs to navigate your vehicle from a starting point to a final destination. Sessions 9 through 11 will guide you through a zigzag route on the logic circuit map to show you the basics. Once you see how easy it is to navigate your car with these basic inputs, you can create your own simulation inputs to complete your challenge of driving to Altera as quickly as possible with the fewest tickets. As in real life, there are many routes to the same destination. It's up to you to find the one that suits you best.

Session 9: Create a Simulator Channel File

In this session, you will learn how to use the MAX+PLUS II Waveform Editor to create and edit simulation input vectors to perform a specific task. It shows how to create and edit a Simulator Channel File (**.scf**) while incorporating some useful command shortcuts.

The **chiptrip.scf** file takes you on a leisurely drive from your company to Altera, via the intersections RPT, MPLD, EPLD, GDF, CNF, and EPM, as shown in blue on the map in Figure 3-6. This session shows you how to adjust your directional and acceleration inputs to achieve the desired outcome.

Figure 3-6. chiptrip.scf Driving Route



Once you practice creating and editing input vectors and simulating **chiptrip.scf**, you will be ready to create your own SCF—**finish.scf**—to drive from your company to Altera as fast as you can while getting as few tickets as possible. The basic steps you learn while creating, editing, and simulating **chiptrip.scf** are the same steps you can use to create **finish.scf**.

If you take a wrong turn or want to take a shortcut, you can copy **chiptrip.scf** or **finish.scf** from the **\max2work\chiptrip** subdirectory into your **\max2work\tutorial** subdirectory. (On a UNIX workstation, the **maxplus2** directory is a subdirectory of the **/usr** directory.)

This session includes the following steps:

1. Create a Simulator Channel File.
2. Add additional node(s) or group(s) to the SCF.
3. Rearrange the order of the nodes and groups.
4. Edit the input node waveforms.
5. Save and close the file.

You can also create simulation inputs in a Vector File (**.vec**) with the MAX+PLUS II Text Editor or another ASCII text editor; however, this procedure is not described in this tutorial. For complete information on Vector Files, go to "Vector File" in MAX+PLUS II Help using **Search for Help on** (Help menu).

1. Create a Simulator Channel File

You can easily create an SCF that contains some or all of the nodes in the Simulator Netlist File (**.snf**) for the compiled project. This "default" SCF can be edited to provide the inputs for simulation.

To create a default SCF:

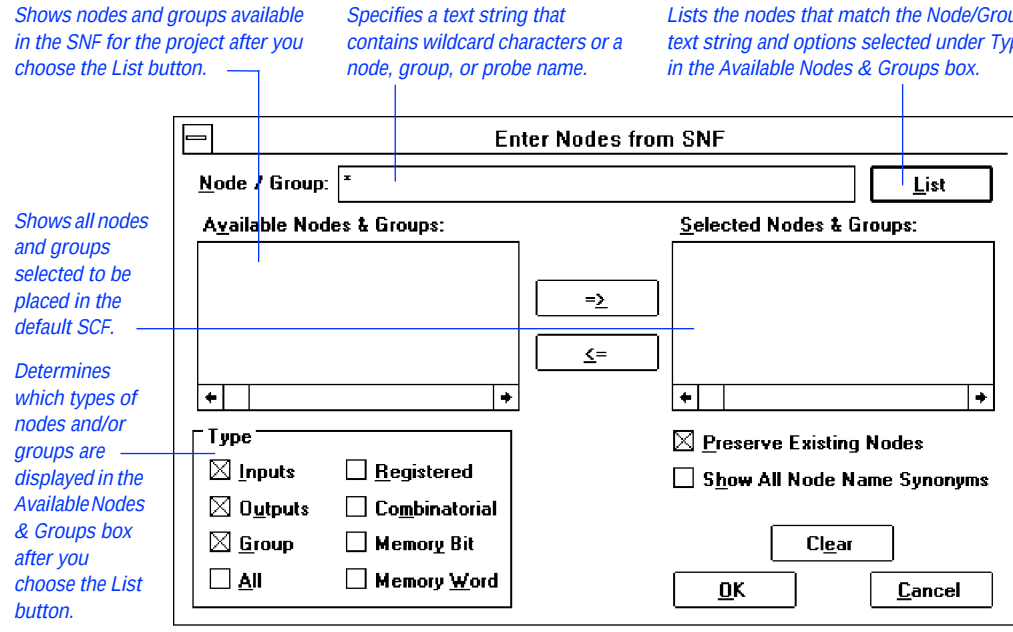
1. Choose **New** from the File menu, select *Waveform Editor file*, select the *.scf* extension in the drop-down list box, and choose **OK** to create a new, untitled file.

2. If necessary, click Button 1 on the **Maximize** button in the Waveform Editor title bar to maximize the window.

3. Choose **End Time** (File menu) and type an end time of `800ns` for the file. The end time determines when the Simulator will stop applying input vectors during simulation.

4. Choose **Grid Size** (Options menu), type `50ns`, and choose **OK**.

5. Choose **Enter Nodes from SNF** (Node menu). The **Enter Nodes from SNF** dialog box is displayed:



SHORTCUTS

Pressing Button 2 in the node/group information area or the waveform drawing area and choosing **Enter Nodes from SNF** from the pop-up menu is a shortcut for opening the **Enter Nodes from SNF** dialog box.

6. Turn off the *Group* option under *Type*. (The *Inputs* and *Outputs* options should remain turned on.)

7. Choose **List** to list the available input (`I`) and output (`O`) nodes.

8. Press Button 1 on the topmost node in the *Available Nodes & Groups* box and drag the mouse down to select the `reset`, `enable`, `dir1`, `dir0`, `clock`, and `accel` input nodes.

9. Choose the right direction button (**=>**) to copy the selected nodes into the *Selected Nodes & Groups* box.

10. Scroll to the end of the list of available nodes to display `at_altera`.

11. Double-click Button 1 on the `at_altera` output node to copy it into the *Selected Nodes & Groups* box.

12. Under *Type*, turn off the *Inputs* and *Outputs* options and turn on the *Group* option.

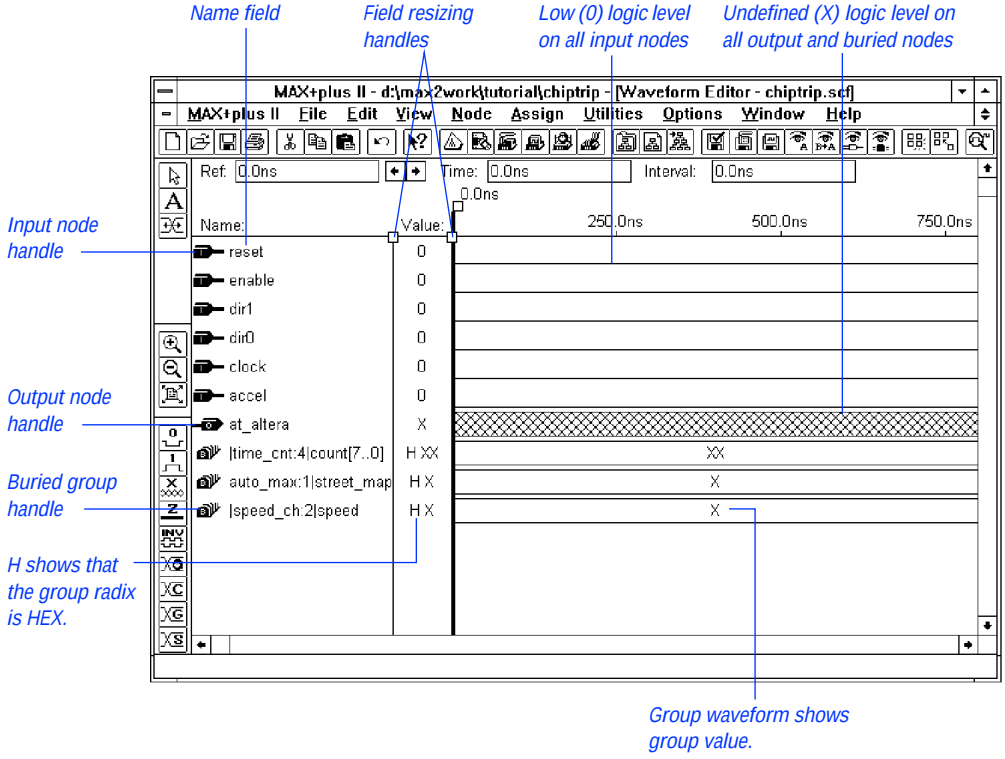
13. Choose **List** to list the available groups.

14. Select the following three buried (`B`) groups in the *Available Nodes & Groups* box: `|time_cnt:4|count[7..0]`, `|auto_max:1|street_map`, and `|speed_ch:2|speed`. You can press Ctrl while clicking Button 1 to select names that are not adjacent to each other in the list.

    ☞ Hierarchical group (and node) names are preceded by a hierarchy path that consists of `|`*<symbol name>*`:`*<symbol ID>*`|`. Your actual symbol ID numbers will vary if you entered symbols in a different order in **chiptrip.gdf**.

15. Choose the right direction button (**=>**) to copy the selected groups into the *Selected Nodes & Groups* box.

16. Choose **OK**. The Waveform Editor overwrites the untitled file with the selected nodes and groups. All input node waveforms have default low (0) logic levels, and all output and buried node waveforms have default undefined (X) logic levels, as shown in the following illustration:



17. (Optional) With Button 1, drag a field resizing handle right or left to change the width of the Name field or Value field.

18. Choose **Save As**. The name **chiptrip.scf** appears automatically in the *File Name* box.

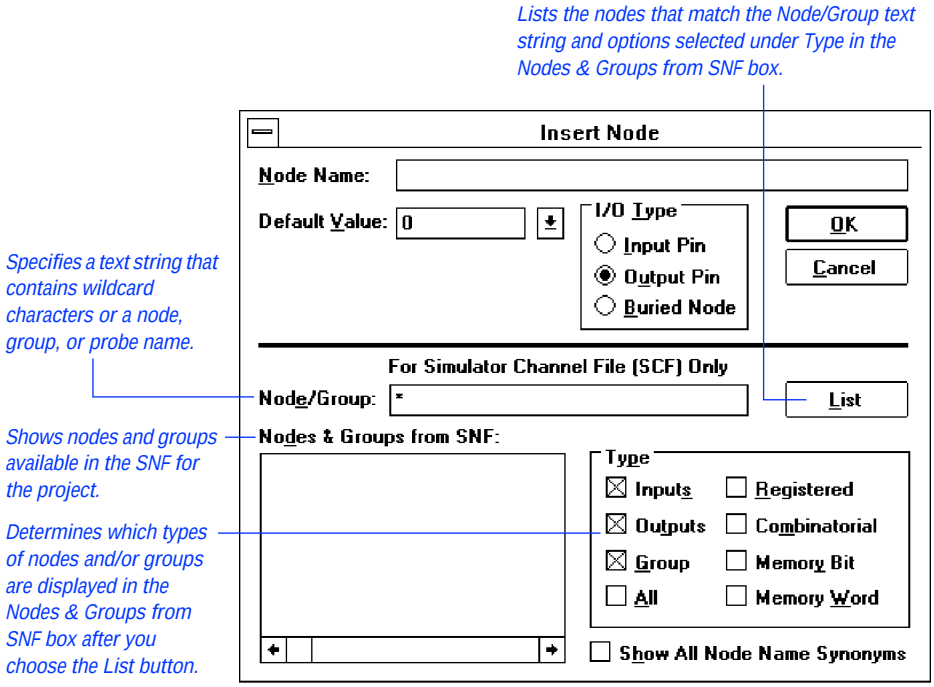
19. Choose **OK** to save the **chiptrip.scf** file.

2. Add Additional Node(s) or Group(s) to the SCF

You can easily add a node or group to your SCF with the **Insert Node** command (Node menu).

To add a node or group:

1. Double-click Button 1 in a blank space in the node/group information area below all existing nodes and groups. The **Insert Node** dialog box is displayed:



SHORTCUTS

Pressing Button 2 anywhere in the node/group information area or the waveform drawing area and choosing **Insert Node** from the pop-up menu is a shortcut for opening the **Insert Node** dialog box.

2. Under *Type,* turn off the *Inputs* and *Outputs* options. The *Group* option should remain turned on.

3. Choose **List** to list the available groups.

4. Select the `ticket[3..0]` output group.

5. Select *X* in the *Default Value* drop-down list box.

6. Choose **OK**. The added group appears in the blank space you selected.

3. Rearrange the Order of the Nodes & Groups

To make the file easier to manage, you can rearrange the nodes and groups in any desired order. For this tutorial, you should move the nodes and groups into the following order: inputs, outputs, and buried logic.

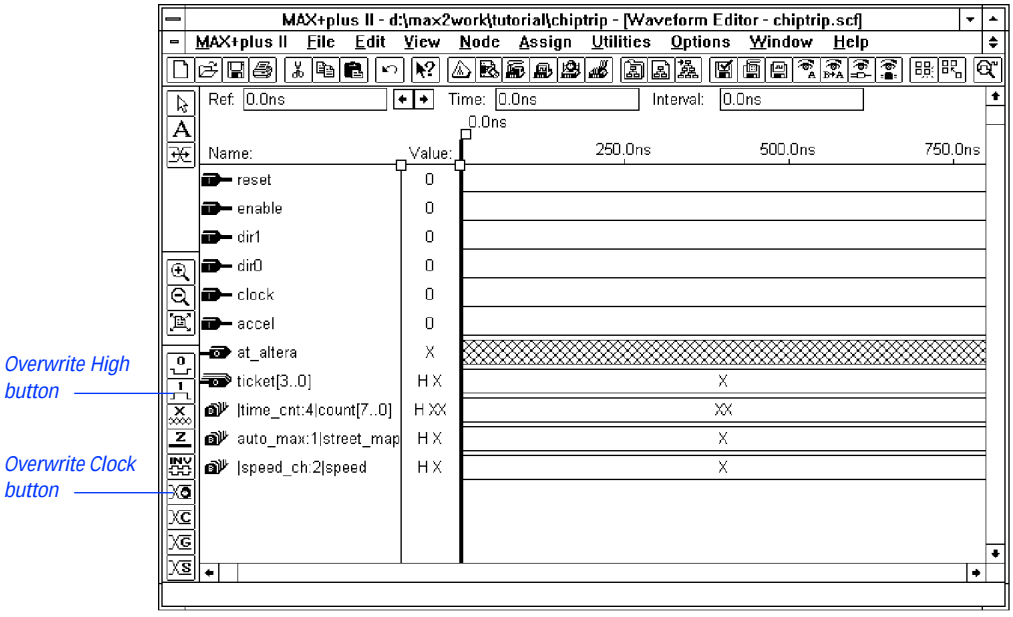
To move the `ticket[3..0]` group:

1. Press Button 1 on the `ticket[3..0]` output group handle, as shown in the illustration on page 249.

2. Drag the mouse up. A horizontal line that represents the moved item(s) shifts up and down as you move the pointer.

3. Move the line between the `at_altera` output node and the `time_cnt:4|count[7..0]` buried group, and release Button 1. The `ticket[3..0]` group moves between them.

See the following illustration:



4. Edit the Input Node Waveforms

You must edit the input waveforms to provide the input vectors for simulation. As you simulate the project, the Simulator automatically overwrites the undefined buried and output node logic levels with outputs that are based on the input node logic levels.

☞ Review "Session 4: Create a Waveform Design File" on page 196 for more information on each of these steps.

To edit the waveforms:

1. With the Selection tool, click Button 1 on the Value field for the `enable` input node and choose **Overwrite High (1)** from the Edit menu to overwrite the entire waveform with a high logic level. A high logic level allows the "clock" in your car to count the Clock pulses required for the vehicle to reach Altera.

SHORTCUTS

Shortcuts for overwriting a high logic level:

- ✓ Select a whole waveform or a waveform interval and choose the **Overwrite High (1)** button from the tool palette on the left side of the Waveform Editor window, as shown in the previous illustration.

or:

- ✓ Press Button 2 on the Value field of a waveform (to select the whole waveform) or on a selected waveform interval and choose **Overwrite High (1)** from the pop-up menu.

2. Overwrite high intervals from 200 to 400 ns on `dir1` and 200 to 500 ns on `dir0`. These input signals provide information to direct your vehicle north, south, east, and north again along the zigzag route to Altera.

3. To create a Clock waveform at the current grid size (50 ns), select the whole `clock` waveform by clicking Button 1 on the Value field, and choose **Overwrite Clock** (Edit menu). The **Overwrite Clock** dialog box is displayed. Choose **OK** to accept the default value.

SHORTCUTS

Shortcuts for creating a Clock waveform:

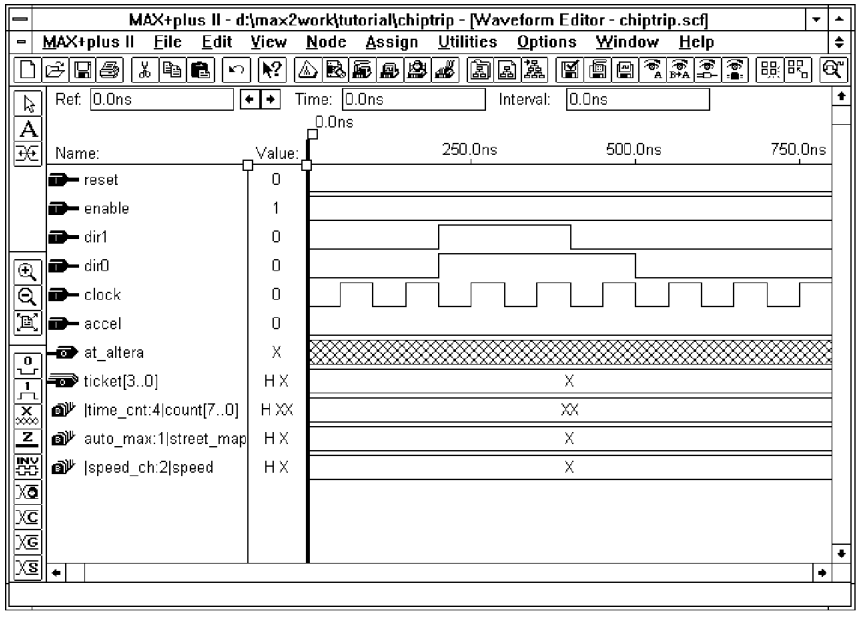
- ✓ Select a whole waveform or a waveform interval and choose the **Overwrite Clock** button from the tool palette on the left side of the Waveform Editor window, as shown in the previous illustration.

or:

- ✓ Press Button 2 on the Value field of a waveform (to select the whole waveform) or on a selected waveform interval and choose **Overwrite Clock** from the pop-up menu.

Both the `reset` and `accel` inputs remain at low (0) logic levels. Since the `accel` signal remains low throughout, your vehicle will not accelerate during the **chiptrip.scf** simulation.

See the following illustration:



5. Save & Close the File

To save and close the file:

1. Choose **Save** (File menu).

2. Choose **Close** (File menu).

☞ If you wish to view the simulated outputs as they are written to the SCF, you should leave the file window open. However, leaving the SCF window open during simulation can reduce the speed of your simulation.

Session 10: Simulate the Project

In this session, you will use the MAX+PLUS II Simulator to simulate the **chiptrip** project. Because the Simulator allows you to verify your project before it is actually committed to hardware, it can dramatically shorten the time it takes to transform your initial design concept into working silicon.

The Simulator uses a Simulator Channel File (**.scf**) or Vector File (**.vec**) as the source of simulation input vectors. In this tutorial, you will use the **chiptrip.scf** file you created in Session 9. This session includes the following steps:

1. Open the Simulator window.
2. Specify additional output files.
3. Turn on setup and hold time monitoring.
4. Run the simulation.
5. Create a Table File.

You can also run the Simulator in batch mode. For complete information on setting up the Simulator to run in batch mode, go to "Running a Batch-Mode Simulation" using **Search for Help on** (Help menu).

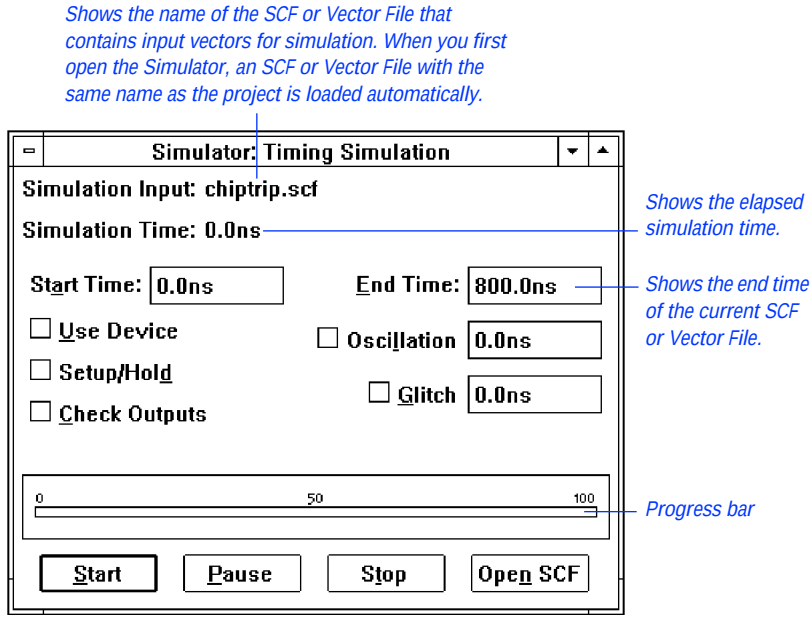
1. Open the Simulator Window

To open the Simulator window:

✓ Choose **Simulator** (MAX+PLUS II menu).

The Simulator Netlist File (**.snf**) for the current project (**chiptrip**) is loaded automatically when you open the Simulator. In addition, **chiptrip.scf**, the Simulator Channel File (**.scf**) created in Session 9, is loaded automatically because it has the same filename as the project.

See the following illustration:

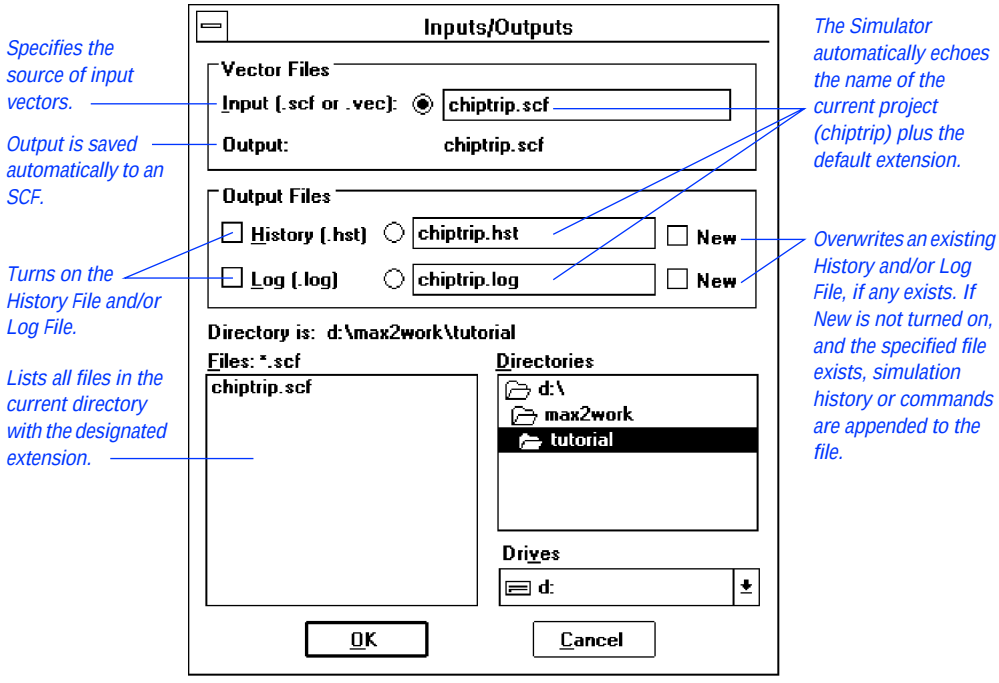


2. Specify Additional Output Files

The **Inputs/Outputs** command (File menu) allows you to specify the file that is the source of simulation input vectors and up to two additional output files: the History File (**.hst**) and Log File (**.log**). The History File records all commands, options, and buttons that are used during a simulation. The command output and all messages generated during simulation are also recorded in this file. The Log File also records the same information, without the command output. You can rename a Log File with the extension **.cmd** and use it as a Command File (**.cmd**) to repeat a simulation in batch mode.

To create History and Log Files:

1. Choose **Inputs/Outputs** (File menu) or double-click Button 1 on the *Simulation Input* field in the Simulator window. The **Inputs/Outputs** dialog box is displayed:



The simulation input file **chiptrip.scf** appears in the *Input (.scf or .vec)* box under *Vector Files*. If you want to use a Vector File or SCF with a name other than the project name, you must specify the filename in this box.

Simulation outputs are automatically saved to an SCF with the same filename as the input file.

2. Turn on the *History (.hst)* and *Log (.log)* options under *Output Files*. The filenames **chiptrip.hst** and **chiptrip.log** appear automatically in the *History (.hst)* and *Log (.log)* boxes.

3. Choose **OK**.

3. Turn On Setup & Hold Time Monitoring

You can monitor the project to determine whether setup and hold time violations occur during simulation.

To turn on setup and hold time monitoring:

✓ Turn on the *Setup/Hold* option in the Simulator window.

4. Run the Simulation

To simulate the project:

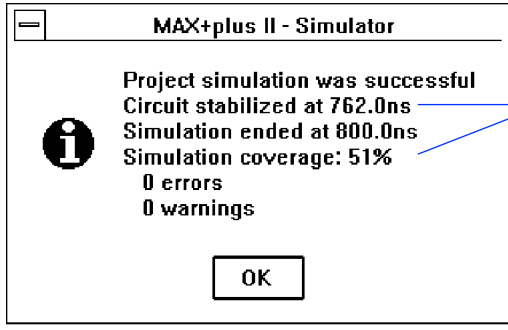
1. Choose the **Start** button.

   The Simulator immediately begins to simulate the **chiptrip** project. As the Simulator processes the input vectors and simulates the project, the progress bar moves toward 100%, the Simulation Time field is dynamically updated, and the output logic levels are recorded in **chiptrip.scf**.

   The Simulator runs in the background, freeing your computer for other work. However, this simulation is short and you will not have to wait long for it to finish. When you analyze a larger, more complex project, you can always start a simulation and then switch to another application to continue your work.

   ☞ If you wish to view the simulation outputs as they are written to **chiptrip.scf**, you can open the file in the Waveform Editor. However, leaving an SCF window open during simulation can reduce the speed of your simulation.

When the Simulator has finished, it displays the following messages in a message box:



2. Choose **OK**.

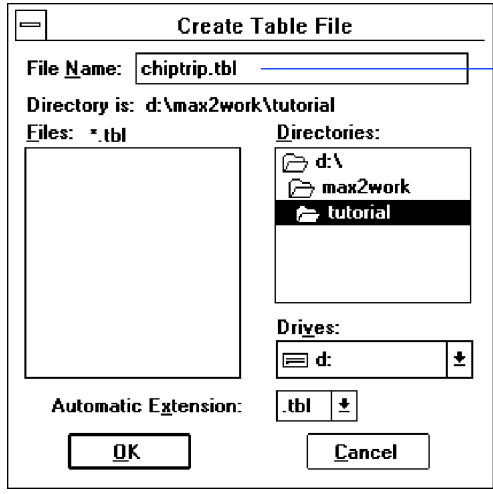
The messages displayed in the message box are also recorded in the History File **chiptrip.hst**. The simulation coverage percentage indicates how many nodes in the project changed logic levels during the simulation. The Simulator also creates the Log File **chiptrip.log**. You can view both files with the MAX+PLUS II Text Editor or another standard text editor. "Session 11: Analyze Simulation Outputs" on page 261 describes how to open and view the History and Log Files.

5. Create a Table File

The Table File (**.tbl**) provides an ASCII alternative to the output in the SCF.

To create a Table File:

1. Choose **Create Table File** (File menu). The **Create Table File** dialog box is displayed:



2. Choose **OK**.

3. The Simulator displays a message stating that the Table File was generated successfully. Choose **OK**.

You can view the Table File with the MAX+PLUS II Text Editor or another standard text editor. "Session 11: Analyze Simulation Outputs," next, describes how to open and view the Table File.

Session 11: Analyze Simulation Outputs

In this session, you will use the MAX+PLUS II Waveform Editor and Text Editor to view the results of simulation. This session includes the following steps:

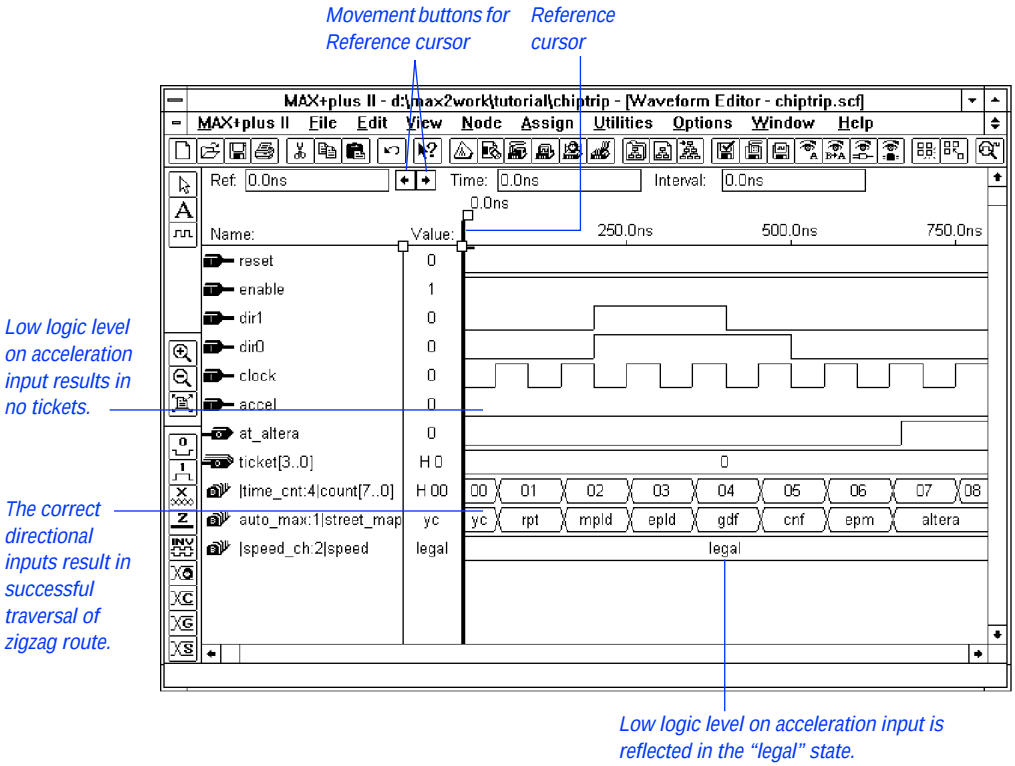
1. View the Simulator Channel File.
2. View the History, Log, and Table Files.
3. Re-edit your SCF if necessary.
4. Create, simulate, and analyze **finish.scf**.

1. View the Simulator Channel File

✓ Choose the **Open SCF** button in the Simulator window to open **chiptrip.scf**, the SCF for the current project.

☞ If you did not close **chiptrip.scf** at the end of Session 9, you can also choose **Waveform Editor** (MAX+PLUS II menu) to bring the most recently active Waveform Editor window to the front.

Now that you have simulated the output node waveforms, their logic levels are defined. The following illustration shows the outputs MAX+PLUS II creates in **chiptrip.scf** after simulation. Try scrolling left and right, or use the **Zoom In** and **Zoom Out** commands (View menu) to examine the file.



Your directional input signals, `dir1` and `dir0`, and the `clock` pulses create the outputs in `auto_max:1|street_map` waveform that represent your vehicle's progress along the zigzag route to Altera. The states on the `street_map` output change at each Clock cycle in the following pattern: `yc`, `rpt`, `mpld`, `epld`, `gdf`, `cnf`, `epm`, and `altera`. (Refer to the map in Figure 3-6 on page 245.) Because the `accel` acceleration input remains low throughout the SCF, the `speed_ch:2|speed` output reflects a `legal` state, and the `ticket[3..0]` group has a low (0) logic level, i.e., no speeding tickets.

To view the exact time at each logic level transition:

1. If necessary, press Button 1 on the Reference cursor handle and drag it to the beginning of the file.

2. Click Button 1 on the right Reference cursor movement button, as shown in the previous illustration, to move the Reference cursor to the first logic level transition in the file.

3. Continue clicking Button 1 on the right Reference cursor movement button to view subsequent logic level transitions.

The time at the Reference cursor location is displayed both at the top of the cursor and in the Reference field. Each transition shows the exact time and location of the signal's change in logic level or state.

2. View the History, Log & Table Files

You can view the History, Log, and Table Files for additional information about your simulation. Use the History File (**.hst**) to review the entire history of your simulation. The Log File (**.log**) is similar to the History File, but without the command outputs; it can be saved as a Command File (**.cmd**) and used to run batch-mode simulation. The Table File (**.tbl**) is a text file that contains the same information as the current SCF or Waveform Design File (**.wdf**). A Table File has the same format as a Vector File (**.vec**).

To view the History, Log, or Table File:

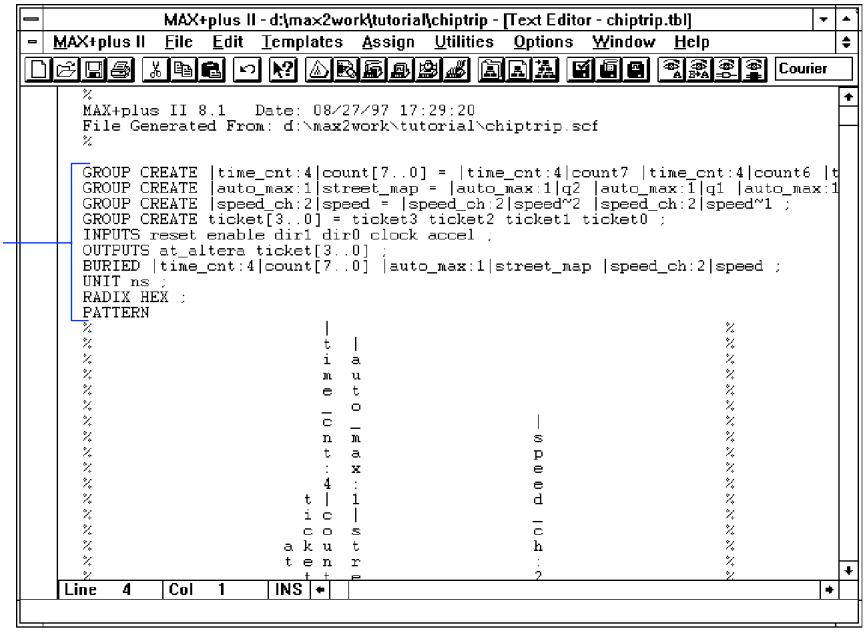
1. Choose **Open** (File menu).

2. Select *Text Editor files* and choose the *.hst*, *.log*, or *.tbl* extension in the drop-down list box.

3. Double-click Button 1 on the appropriate **chiptrip** file in the *Files* box.

MAX+PLUS II automatically opens the Text Editor window and displays the file you selected.

The following illustration shows the Table File **chiptrip.tbl**:



Simulation results are saved in Vector File format.

3. Re-Edit Your SCF if Necessary

You may need to edit your inputs and simulate your project repeatedly to obtain the desired simulation results. Even very experienced digital designers simulate their designs many times with multiple sets of inputs. MAX+PLUS II allows you to quickly complete many different simulation runs to verify the functionality of your logic circuit design. You should use the Simulator to exhaustively test your project before you program it into a device.

4. Create, Simulate & Analyze finish.scf

To practice your new skills, return to "Session 9: Create a Simulator Channel File" on page 245 and create another SCF called **finish.scf**. Your challenge will be to plot a path on the logic circuit map shown on page 245 that takes you from your company to Altera as quickly as possible with the fewest speeding tickets.

Because **finish.scf** contains the same nodes and groups as **chiptrip.scf**, you can use **Save As** (File menu) to save **chiptrip.scf** as **finish.scf**, and proceed to edit the input nodes. As a shortcut, you can also copy **finish.scf** from the **\max2work\chiptrip** subdirectory.

Once you successfully drive from your company to Altera by creating and simulating **finish.scf**, you might try one of the following challenges, or plot your own itinerary:

- Travel from your company to Altera as quickly as possible, passing through all intersections, with as few tickets as possible.
- Travel from your company to Altera as quickly as possible, regardless of tickets.

Session 12: Analyze Timing

In this session, you will use the Timing Analyzer to analyze the performance of the **chiptrip** project. The Timing Analyzer offers three analysis modes:

Analysis Mode:	Description:
Delay Matrix	Analyzes the propagation delay paths between multiple source and destination nodes.
Registered Performance	Analyzes registered logic for a performance-limiting delay, minimum Clock period, and maximum circuit frequency.
Setup/Hold Matrix	Calculates the minimum setup and hold time requirements from input pins to signal inputs of flipflops, latches, and asynchronous RAM.

You will use the timing Simulator Netlist File (**.snf**) generated in "Session 6: Compile the Project" on page 216 to analyze the propagation delays in the **chiptrip** project. This session includes the following steps:

1. Open the Timing Analyzer window.
2. Run the Timing Analyzer.
3. List a propagation delay message.
4. Locate the delay path in the Floorplan Editor.
5. Locate the delay path in the project's design files.
6. Run a timing analysis in another mode.

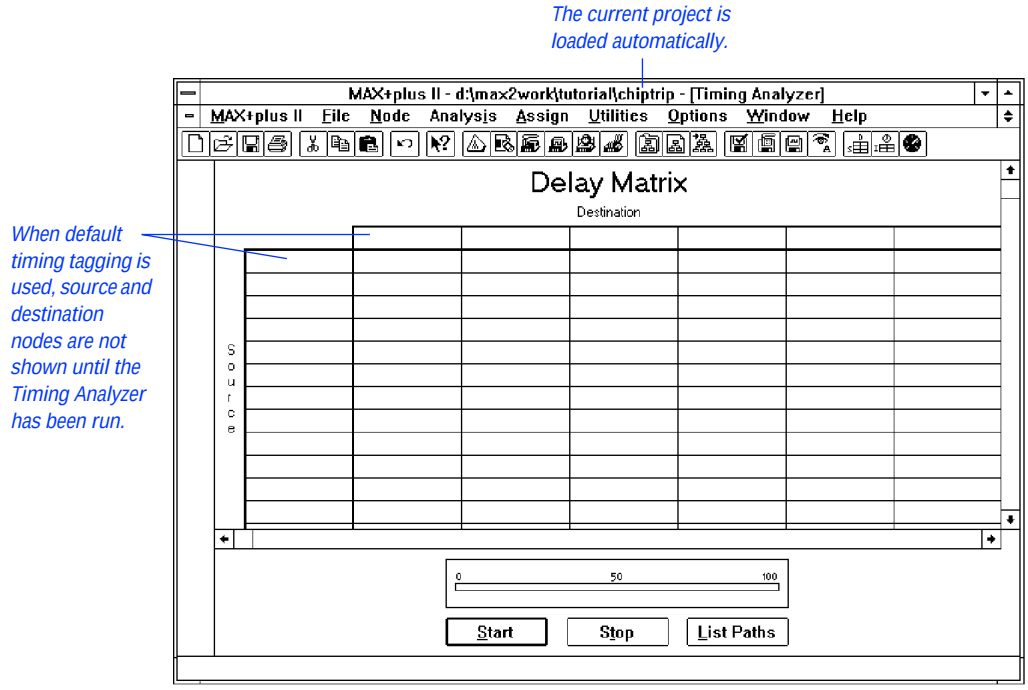
1. Open the Timing Analyzer Window

To open the Timing Analyzer window:

1. Choose **Timing Analyzer** (MAX+PLUS II menu).
2. If necessary, click Button 1 on the **Maximize** button in the title bar to maximize the window.
3. If the Delay Matrix is not already displayed, choose **Delay Matrix** (Analysis menu).

The Timing Analyzer automatically loads the timing SNF for the **chiptrip** project.

See the following illustration:



The Timing Analyzer automatically "tags" all input pins as timing sources and all output pins as timing destinations for Delay Matrix analysis. The names of nodes with this "default timing analysis tagging" are not visible until after the analysis is run. Each of the three analysis modes has its own display and appropriate default timing analysis tagging for nodes.

You can tag specific nodes for analysis in the Timing Analyzer; in the Floorplan Editor; or in the original project design files in the Graphic, Text, and Waveform Editors. The **Timing Analysis Source** and **Timing Analysis Destination** commands are provided in all of these MAX+PLUS II applications.

Choose the context-sensitive help button  and click Button 1 anywhere within the matrix to go to "Delay Matrix Display" in MAX+PLUS II Help.

2. Run the Timing Analyzer

To run a timing analysis:

1. Turn on the **Cut Off I/O Pin Feedback** command (Options menu). When this command is turned on, the Timing Analyzer uses bidirectional I/O pins only as source and destination nodes; feedback from within the device is excluded.

2. Turn off the **Cut Off Clear & Preset Paths** command (Options menu). When this command is turned off, the Timing Analyzer calculates paths that travel through the Clear and Preset inputs to D flipflops. If a design does not use Clear or Preset signals, or if you do not wish to display paths that travel through Clear and Preset inputs to D flipflops, you can turn this command on.

3. Choose **Start**. The Timing Analyzer immediately begins to analyze the **chiptrip** project and calculate the minimum and maximum propagation delays between each pair of nodes that are connected in the project. The progress bar moves toward 100%.

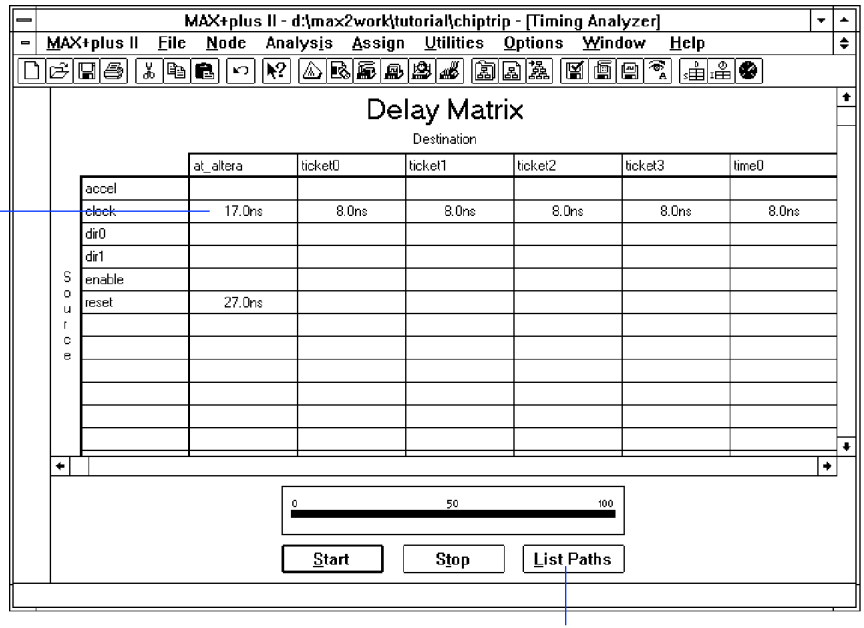
   

   You can also run a timing analysis by choosing **Analyze Timing** (Utilities menu) in the Graphic, Text, Waveform, or Floorplan Editor.

The Timing Analyzer runs in the background, freeing your computer for other work. However, this analysis is short and you will not have to wait long for it to finish. When you analyze a larger, more complex project, you can start an analysis and then switch to another application to continue your work.

4. When the message `Timing analysis is completed` is displayed, choose **OK**.

The path between each pair of nodes is displayed in a cell in the Delay Matrix, as shown in the following illustration:



Delay times may differ from those in the illustration shown above when you analyze the project. Timing Analyzer results are based on the latest device performance data given in the Device Model Files (**.dmf**) provided with MAX+PLUS II.

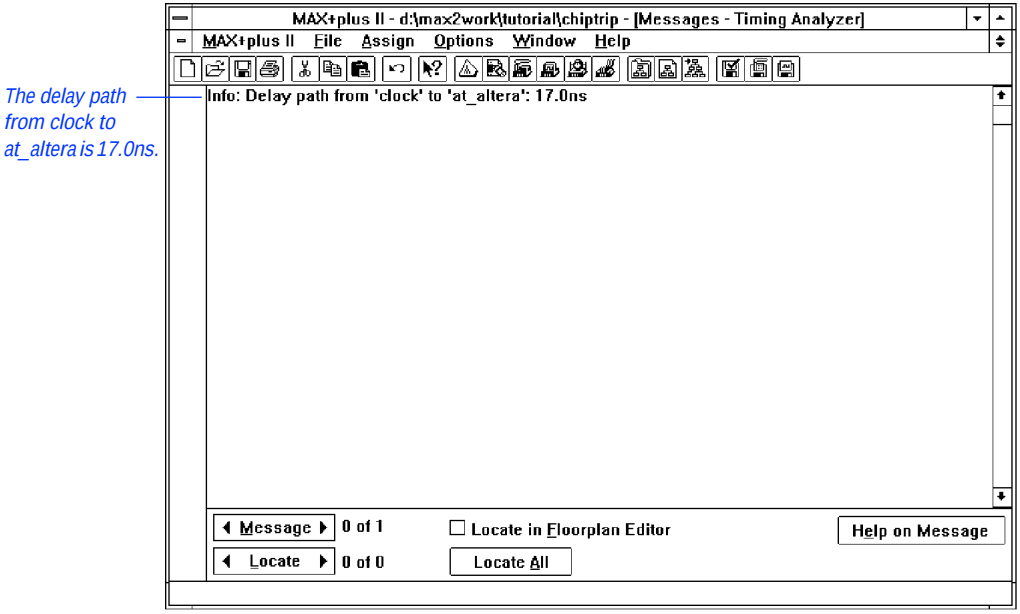
If the shortest and longest routes through the project are different, both delay times are displayed in a cell. If the two figures are different, the circuit contains a potential logic race condition. When source and destination nodes are separated by the D input to a flipflop in the original design file, the delay is calculated through the Clock or Preset input, not the D input.

3. List a Propagation Delay Message

To list the delays for the signal paths represented by a cell:

1. Select the cell for `clock` and `at_altera`.
2. Turn on the **List Only Longest Path** command (Options menu). When this command is turned on before you choose the **List Paths** button, the Timing Analyzer displays only the longest delay path in the Message Processor window.
3. Choose the **List Paths** button. The Message Processor window opens and lists the longest delay path between the `clock` and `at_altera` nodes.
4. The message `Finished listing longest delay path(s)` is displayed. Choose **OK**.
5. If necessary, choose **Message Processor** (MAX+PLUS II menu) to bring the Message Processor window to the front.

See the following illustration:



4. Locate the Delay Path in the Floorplan Editor

You can direct the Message Processor to locate the path represented by a message in the Floorplan Editor. You can either locate each source of a message in succession, or locate all sources simultaneously.

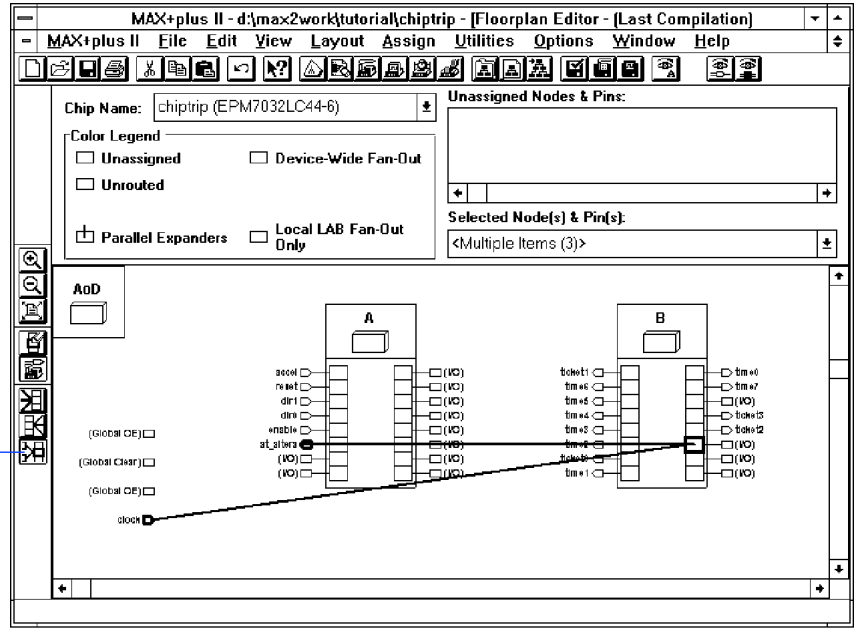
To locate all sources of a message simultaneously in the Floorplan Editor:

1. In the Message Processor window, click Button 1 on the message or on the right side of the **Message** button to select the message `Info: Delay path from 'clock' to 'at_altera': 17.0ns`.

2. Turn on the *Locate in Floorplan Editor* option. The **Locate All** button becomes active (i.e., undimmed).

3. Choose the **Locate All** button. MAX+PLUS II automatically opens the Floorplan Editor window and highlights all sources of the message simultaneously.

4. If necessary, turn on the **LAB View** and **Last Compilation Floorplan** commands (Layout menu) and the **Show Path** command (Options menu); and turn off the **Report File Equation Viewer** command (Layout menu). The full path from the `clock` source node to the `at_altera` destination node is displayed, as shown in the following illustration:



5. Once you finish viewing the Floorplan Editor, close the Floorplan Editor window and return to the Timing Analyzer.

5. Locate the Delay Path in the Project's Design Files

You can direct the Message Processor to locate the path represented by a message in the original design file(s) for a project.

To locate the sources of a the message in the **chiptrip** project's design files:

1. In the Message Processor window, click Button 1 on the message or on the right side of the **Message** button to select the message `Info: Delay path from 'clock' to 'at_altera': 17.0ns`.

2. Turn off the *Locate in Floorplan Editor* option.

3. Click Button 1 on the right side of the **Locate** button to locate the first of four sources for the message. MAX+PLUS II automatically opens the file **chiptrip.gdf** in a Graphic Editor window and highlights the `clock` input pin.

4. Locate each of the other three sources by clicking Button 1 on the right side of the **Locate** button. MAX+PLUS II automatically opens the appropriate design editor for each successive message source.

You can close any open editor window(s) and the Message Processor and return to the Timing Analyzer to select other cells, list delay path messages, and locate the paths in either the Floorplan Editor or the source design files for the project.

6. Run a Timing Analysis in Another Mode

✓ Return to the Timing Analyzer window, choose **Registered Performance** or **Setup/Hold Matrix** (Analysis menu), and choose **Start** to run the Timing Analyzer again.

Session 13: Program an Altera Device

In this session, you will use the MAX+PLUS II Programmer to program the **chiptrip** project into an Altera EPM7032LC44 device (a 44-pin plastic J-lead chip carrier package), which was selected automatically during compilation. This session includes the following steps:

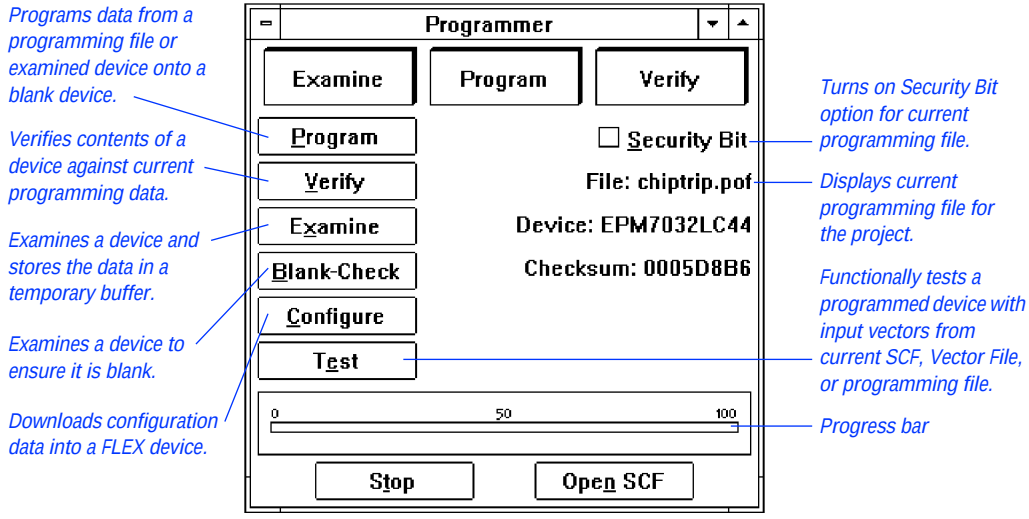
1. Open the Programmer window.
2. Create an output Programmer Log File.
3. Program the device.

1. Open the Programmer Window

Make sure that the programming hardware is installed. See "Installing the Programming Hardware" on page 53 in *MAX+PLUS II Installation* for more information.

To open the Programmer window:

✓ Choose **Programmer** (MAX+PLUS II menu). The Programmer window opens, as shown in the following illustration:



The File field displays the current POF, **chiptrip.pof**.

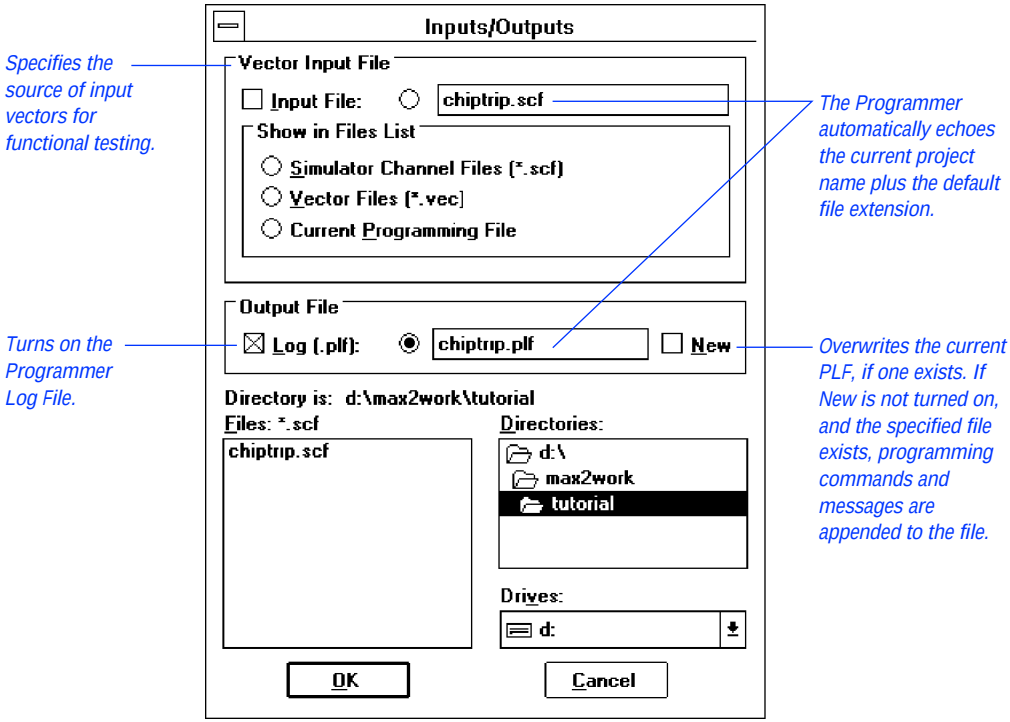
If **chiptrip.pof** is not shown, use the **Select Programming File** command (File menu) to select **chiptrip.pof** as your programming file. You will be asked whether you wish to change the current project to **chiptrip**. Choose **OK**.

2. Create an Output Programmer Log File

MAX+PLUS II optionally records all Programmer actions and messages in a Programmer Log File (**.plf**).

To create an output PLF:

1. Choose **Inputs/Outputs** (File menu). The **Inputs/Outputs** dialog box is displayed:



2. If necessary, turn on the *Log (.plf)* option under *Output File*. The filename `chiptrip.plf` appears automatically in the *Log (.plf)* box.

3. Choose **OK**.

Press F1 when the **Inputs/Outputs** dialog box is displayed to go to "Inputs/Outputs Command" in MAX+PLUS II Help.

3. Program the Device

To program the device:

1. Insert an EPM7032LC44-6 device into the programming socket.

2. Choose the **Program** button.

   The Programmer examines the device, programs the **chiptrip** project into the device, and checks that the contents of the device match those of the **chiptrip.pof** file. Once programming is complete, you can view the PLF if you wish. For more information on how to open and view text files, see "2. View the History, Log & Table Files" on page 263.

3. Double-click Button 1 on the document icon (or box) to close the Programmer window.

Go to "Inserting a Device into the Socket" and "Programming a Single Device with the Master Programming Unit" using **Search for Help on** (Help menu).

Are We There Yet?

Congratulations on finishing the tutorial! If you would like to further practice what you have learned, you can return to selected sessions and try the following:

- Try creating the subdesigns with different design entry methods, e.g., create the **speed_ch** subdesign as a Text Design File (**.tdf**).
- Compile the **chiptrip** project for different devices. For example, try the EPM9320 device and run a new timing analysis.
- Compile and simulate the project for different speed grades of the same device.
- Run timing analyses for the new devices.
- If you recompile the project for a different device, try programming the actual device.

Appendix

A

MAX+PLUS II Command-Line Mode

You can operate the MAX+PLUS II Compiler, Timing Analyzer, and Simulator from the command prompt under UNIX, Microsoft Windows NT, and Microsoft Windows 95.

To run MAX+PLUS II from a command prompt, type:

`maxplus2 -h | -v | {` *<batch option(s)>* `[` *<I/O option(s)>* `]` *<project name>* `}` ↵

Multiple batch and I/O options can be used for a single project; multiple projects can be processed with the same command line. The *<project name>* indicates the end of the options for that project.

The *<batch options>* are as follows:

Batch Option:	**Action:**
`-h` *or* `-help`	displays information about command line options
`-v` *or* `-version`	displays the MAX+PLUS II version number
`-c` *or* `-compile`	runs the Compiler
`-ta_delay`	runs the Timing Analyzer in Delay Matrix mode
`-ta_setup`	runs the Timing Analyzer in Setup/Hold Matrix mode
`-ta_reg`	runs the Timing Analyzer in Registered Performance mode
`-s` *or* `-simulate`	runs the Simulator
`-i` *or* `-ignore_errors`	ignores errors from the Compiler, Simulator, or Timing Analyzer, and continues processing other projects specified in the same command line, even if the processing on a previous project has failed

The *<I/O options>* are shown below. For each option, the *<filename>* defaults to *<project name>* if you specify empty quotation marks (`""`).

I/O Option:	Action:
`-tao "`*<filename>*`"`	saves Timing Analyzer output in *<filename>*.**tao**; if this option is not used, *<project name>*.**tao** is generated automatically
`-scf "`*<filename>*`"`	uses *<filename>*.**scf** as the source of simulation vectors; if this option is not used, the file specified with the `SIMULATION_INPUT_FILE` variable in *<project name>*.**acf** is used automatically
`-vec "`*<filename>*`"`	uses *<filename>*.**vec** as the source of simulation vectors
`-cmd "`*<filename>*`"`	uses *<filename>*.**cmd** file as the source of simulation commands
`-tbl "`*<filename>*`"`	saves Simulator output in *<filename>*.**tbl**
`-hst "`*<filename>*`"`	records Simulator history in *<filename>*.**hst**

You can use **>** or **>>** to redirect MAX+PLUS II warning and error messages to an ASCII file. (In UNIX, use **>!** and **>>!** instead of **>** and **>>** if the `noclobber` variable is set.)

The following example compiles the **upcntr** project; compiles the **chiptrip** project; runs a timing analysis on **chiptrip** in Registered Performance mode; and simulates **chiptrip** using **test.scf** as the source of vectors:

```
maxplus2 -c upcntr -c -ta_reg -s -scf "test.scf" chiptrip ↵
```

The following example compiles the **chiptrip** project, overwriting any existing **message.out** file:

```
maxplus2 -c chiptrip > message.out ↵
```

Appendix

B

Altera Support Services

Altera's support team is dedicated to resolving your technical issues quickly. Altera responds to your questions promptly and efficiently via telephone, fax, or e-mail. Applications Engineers are located at Altera headquarters in San Jose, California, and at locations around the world.

The Altera Applications, Literature, and Marketing Departments offer the following services:

- Product information
- Technical support
- Technical publications
- Training courses

Contacting Altera Support Services

Table B-1 describes Altera's support services.

Table B-1. Altera Support Services (Part 1 of 2)

Support Service	Contact Information *Note (1)*	Description
Product Information	Tel: (408) 544-7104 E-mail: **news@altera.com** WWW: **http://www.altera.com** BBS: 544-6421 *Note (2)* FTP site: **ftp@altera.com** *or* contact your local Altera distributor or sales office	Up-to-date information on Altera products is available from the Altera Marketing Department between the hours of 8:00 a.m. and 5:00 p.m. Pacific Time, Monday through Friday.
Technical Support	**U.S. & Canada Only:** Hotline: (800) 800-EPLD *or* (408) 544-7000 **Worldwide:** Tel: (408) 544-7000 Fax: (408) 544-6401 WWW: **http://www.altera.com** BBS: 544-6421 *Note (2)* FTP site: **ftp@altera.com** E-mail: **sos@altera.com** Contact your local Altera distributor or sales office for design evaluations and on-site support	Direct technical support on Altera devices and software is available from the Altera Applications Department between the hours of 6:00 a.m. and 6:00 p.m. Pacific Time, Monday through Friday. Applications Engineers at Altera and Field Applications Engineers located around the world can evaluate customer designs, recommend efficient design methods and devices that will best meet your needs, estimate device performance, demonstrate MAX+PLUS II software, and provide on-site training. The world-wide web (WWW) site provides access to the Atlas technical support database, and to product information and technical publications. You can use the FTP site and the BBS to transfer files to and from the Altera Applications Department for technical support and review. The FTP site and the BBS also provide software utilities and technical publications.

Notes:
(1) Altera's e-mail, world-wide web (WWW) site, bulletin board service (BBS), and File Transfer Protocol (FTP) site are available 24 hours a day.
(2) The BBS requires a Bell Standard 212, CCITT standard, or compatible modem at up to 14,400 bps, using 8 data bits, 1 stop bit, and no parity.

Table B-1. Altera Support Services (Part 2 of 2)

Support Service	Contact Information *Note (1)*	Description
Technical Publications	Tel: (888) 3-ALTERA E-mail: **lit_req@altera.com** WWW: **http://www.altera.com** BBS: 544-6421 *Note (2)* FTP site: **ftp@altera.com**	Altera produces a variety of technical literature to help you select and design with programmable logic, including application notes and data sheets. Altera also provides *News & Views*, a quarterly newsletter that includes the latest information on Altera products, technical articles written by Altera Applications Engineers, and a question and answer section that addresses many commonly asked questions. All registered users of Altera products receive *News & Views*.
Training Courses	Altera Training Administrator: Tel: 544-7000 *or* contact your local Altera sales office	Altera provides a variety of training courses to teach you innovative and efficient design techniques. With these courses, you can discover the time-saving features of the MAX+PLUS II development system, explore the design features of Altera's device families, or simply learn about Altera products.
Notes: (1) Altera's e-mail, world-wide web (WWW) site, bulletin board service (BBS), and File Transfer Protocol (FTP) site are available 24 hours a day. (2) The BBS requires a Bell Standard 212, CCITT standard, or compatible modem at up to 14,400 bps, using 8 data bits, 1 stop bit, and no parity.		

Go to "Contacting Altera" in MAX+PLUS II Help for up-to-date information on Altera contact information.

Appendix

C

Additional Workstation Configuration Information

This section describes how to change additional workstation configuration items that control the appearance of MAX+PLUS II windows, serial port configuration, screen height and width, printer ports, and fonts.

Customizing MAX+PLUS II Colors

You can customize the colors of various elements in the MAX+PLUS II window by editing the ASCII-format **win.ini** file, which is copied into the ***<user's home directory>*/windows** directory the first time you run MAX+PLUS II.

The settings in this file determine the colors of basic window elements when Windows "look and feel" is selected. For more information about changing the "look and feel" of the user interface, go to "Environment Variables" on page 288. In contrast, the **Color Palette** command (Options menu) in MAX+PLUS II determines the colors of specific objects displayed in individual MAX+PLUS II application windows.

The `[colors]` section of **win.ini** defines the color of various elements in the MAX+PLUS II window. Three values in the range of 0 to 255 define the amount of red, green, and blue (RGB) that determine the color of each element.

The following table shows the `[colors]` section of a sample **win.ini** file and a brief description of each window element.

Component = R G B Value:	Description:
`Background=192 192 192`	Desktop background
`AppWorkspace=255 255 255`	MAX+PLUS II workspace
`Window=255 255 255`	MAX+PLUS II application workspace
`WindowText=0 0 0`	Window text
`Menu=255 255 255`	Window background
`MenuText=0 0 0`	Menu text
`ActiveTitle=0 0 128`	Active window title bar
`InactiveTitle=255 255 255`	Inactive window title bar
`TitleText=255 255 255`	Title bar text in an active window
`ActiveBorder=192 192 192`	Active window border
`InactiveBorder=192 192 192`	Inactive window border
`WindowFrame=0 0 0`	Window frame
`ScrollBar=192 192 192`	Scroll bar background
`ButtonFace=192 192 192`	Button front surface
`ButtonShadow=128 128 128`	Shadow (i.e., darker edges) of an unpressed button

Component = R G B Value:	Description:
`ButtonText=0 0 0`	Text on the face of a button
`GrayText=128 128 128`	Text color when a menu command is unavailable
`Hilight=0 0 128`	Background behind highlighted text
`HilightText=255 255 255`	Highlighted text
`InactiveTitleText=0 0 0`	Text in the title bar of an inactive window
`ButtonHilight=255 255 255`	Lighter edges of an unpressed button

The following table shows the RGB values of the 16 standard colors normally available on a color monitor. You can edit the colors in the `[colors]` section of your **win.ini** file using these values to change the color of the various window components. The availability of other colors depends on the capabilities of your workstation's display system.

Color:	Red:	Green:	Blue:
White	255	255	255
Light Gray	192	192	192
Dark Gray	128	128	128
Black	0	0	0
Red	255	0	0
Dark Red	128	0	0
Green	0	255	0
Dark Green	0	128	0
Blue	0	0	255
Dark Blue	0	0	128
Yellow	255	255	0
Dark Yellow	128	128	0
Magenta	255	0	255
Dark Magenta	128	0	128
Cyan	0	255	255
Dark Cyan	0	128	128

If the appearance of colors in MAX+PLUS II is not satisfactory, and editing the **win.ini** file does not help the problem, you should select the Windows "look and feel," either by setting the `MWLOOK` environment variable described on page 290, or with the Change Look system menu.

Using the mwcolormanager Utility

Altera provides the **mwcolormanager** utility to correct color flickering problems that might occur when you change to MAX+PLUS II from another application window. If another application changes the system colors after you have started MAX+PLUS II, the colors of MAX+PLUS II window elements may change when you return to MAX+PLUS II.

To correct color flickering problems when changing applications, insert the following line as the first command in your **.xinitrc** file:

`mwcolormanager [-display` *<display>*`][-extra` *<nn>*`]↵`

The `-display` *<display>* option allows you to specify a different display than the default listed in the `DISPLAY` variable in your **.cshrc** (C shell users) or **.profile** (Bourne or Korn shell users) file.

The `-extra` *<nn>* option allows you to specify a number *nn* of colors in addition to the 20 colors that are allocated by default. Because MAX+PLUS II only uses 16 colors, you should not use this option.

Environment Variables

MAX+PLUS II uses environment variables to configure various options and locate its files. MAX+PLUS II initializes them when it is installed, but you may wish to change them to optimize your system performance.

If you are using the C shell, environment variables are located in your **.cshrc** file, and have the following format:

`setenv` *<environment variable>* *<value>*

If you are using the Bourne or Korn Shell, environment variables are located in your **.profile** file, and have the following format:

`set` *<environment variable>*=*<value>*

MAX2_HOME

The `MAX2_HOME` variable specifies the name of the MAX+PLUS II home directory. The default is **/usr/maxplus2**. You should use this variable only if the system displays an error message indicating that MAX+PLUS II files cannot be found when you start the program.

MAX2_PLATFORM

The `MAX2_PLATFORM` variable specifies the name of the platform used to run MAX+PLUS II. You should use this variable only if the following error message is displayed when you start the program: **Unable to determine the type of system you are using**.

The following table lists the supported MAX+PLUS II platform names and corresponding variable values:

Platform Name:	Variable Value:
SPARCstation running SunOS 4.1.3+	`sunos`
SPARCstation running Solaris 2.5+	`solaris`
HP 9000 Series 700/800	`hp`
IBM RISC System/6000	`rs6000`

MWCOM1, MWCOM2, MWCOM3 & MWCOM4

These variables control the mapping of serial ports in MAX+PLUS II, which MAX+PLUS II accesses by the names COM1 through COM4, to the corresponding UNIX serial tty ports. Table C-1 shows the default variable values.

Table C-1. Serial Ports

Platform Name	MWCOM1	MWCOM2	MWCOM3	MWCOM4
IBM RISC System/6000	**/dev/tty0**	**/dev/tty1**	**/dev/tty1**	**/dev/tty1**
SPARCstation running SunOS 4.1.3	**/dev/ttya**	**/dev/ttyb**	**/dev/ttyc**	**/dev/ttyd**
SPARCstation running Solaris 2.5	**dev/term/0**	**dev/term/1**	**dev/term/2**	**dev/term/3**
HP 9000 Series 700/800	**/dev/ttyd00**	**/dev/ttyd01**	**/dev/ttyd02**	**/dev/ttyd03**

You can change the default mapping to reassign a COM port to one of the UNIX serial ports. For example, `MWCOM1=/dev/ttyc` binds the **ttyc** serial port to COM1, replacing the default port **ttya**.

MWFONT_CACHE_DIR

The `MWFONT_CACHE_DIR` variable specifies the name of the MAX+PLUS II font cache directory. The default directory is **/*<user's home directory>*/windows**.

MWLOOK

The `MWLOOK` variable controls the initial "look and feel" of the MAX+PLUS II software on the workstation. `MWLOOK` may take the following values:

Value:	Effect:
`motif`	OSF/Motif look and feel
`windows`	Microsoft Windows look and feel

The default value for `MWLOOK` is `windows`.

MWRGB_DB

The MWRGB_DB variable specifies the full pathname to the file **rgb.txt**, which maps color names to 24-bit RGB color values in the X server. If MWRGB_DB is not used, the program looks for **rgb.txt** in the following directories, in order:

1. **/usr/openwin/lib**
2. **/lib**
3. **/usr/X/lib**
4. **/usr/lib/X11**

MWSCREEN_HEIGHT & MWSCREEN_WIDTH

The MWSCREEN_HEIGHT and MWSCREEN_WIDTH variables control the literal size of objects on the screen. They can be set to the actual screen height and width of your display, in millimeters. The default values are those of the X server.

MWSYSTEM_FONT

The MWSYSTEM_FONT variable specifies the default system font used by MAX+PLUS II. If this variable is not used, the default font is Helvetica. To change the system font, set this variable to an existing X font name. For more information, see "Fonts" on page 292.

MWUNIX_SHARED_MEMORY

The MWUNIX_SHARED_MEMORY variable determines whether or not MAX+PLUS II may use UNIX shared memory when it shares data with another program.

If MWUNIX_SHARED_MEMORY is set to `true`, MAX+PLUS II can use UNIX shared memory, which may improve its speed performance. If it is set to `false` (the default value), MAX+PLUS II uses shared memory in the X server when it shares data with another program. This shared memory allows data exchanges between programs that run on different machines, but which are displayed on the same X server.

MWWM

The MWWM variable determines which window manager is used on the system. MWWM may take the following values:

Value:	Effect:
MWM	uses Motif as the window manager
OLWM	uses OpenLook as the window manager
TWM	uses the standard X window manager

MAX+PLUS II automatically detects whether you are using Motif or OpenLook. This variable should be used only if you are using the standard X window manager, TWM.

Fonts

MAX+PLUS II installs the fonts necessary for normal operation. By default, these fonts are located in the **/*<user's home directory>*/maxplus2/fonts** directory.

Adding New Fonts

You can add new X fonts to your system by performing the following steps:

✓ Copy the fonts into the X11 font directory (**/usr/lib/x11/fonts** by default) and restart the X server.

or:

1. Copy the new fonts into the **/usr/maxplus2/fonts** directory.
2. Make a font directory file (**font.dir**) by running the **mkfontdir** utility (**bldfamily** utility on SunOS).
3. Type `xset +fp /usr/maxplus2/fonts` ↵ to prepend the **font.dir** file to the front of the existing font path.

4. Type `xset fp rehash` ↵ to reinitialize the font cache in the X server.

5. Depending on your operating system, perform one of the following:

   ✓ For Solaris, HP-UX, and AIX with Common Desktop Environment (CDE) 1.0 (autostart enabled), go through the following steps:

      a. Add the line `DTSOURCEPROFILE=true` to the **.dtprofile** file in your home directory.

      b. Edit the **.login** file in your home directory to read as follows (the following example is for C shell users):

```
if(!$?DT) then
<all text from your original .login file>
else
/usr/maxplus2/bin/mwcolormanager &
xset fp+ /usr/maxplus2/fonts
xset fp rehash
endif
```

or:

   ✓ For SunOS and Solaris (without Common Desktop Environment), add the following line to the **.xinitrc** file in your home directory to add the directory to your font path each time the X server is started:

```
xset fp+ /usr/maxplus2/fonts
```

A system default version of the **.xinitrc** file, called `Xinitrc`, is available in the `/usr/openwin/lib/` directory.

Font Aliases

The [FontSubstitutes] section of the **win.ini** file in the ***<user's home directory>*/windows** directory provides aliases for font names. These aliases are used to bind the MAX+PLUS II font names to the font names available under the X server.

The following example shows the default font substitution list:

```
[FontSubstitutes]
Helv=helvetica
MS Sans Serif=ms sans serif
Tms Rmn=times
MS Serif=times
Times New Roman=times
Arial=helvetica
```

Printers

MAX+PLUS II uses its own Postscript printer driver to support Postscript printers under UNIX.

Installing a New Printer

The following examples show how to edit the various sections of the **win.ini** file to install a new printer.

```
[windows]
device=Apple LaserWriter II NT,PSCRIPT,LPT1
...
```

The device variable in the [windows] section defines the default printer using the following syntax:

device=*<output device name>*,*<device driver>*,*<port connection>*

MAX+PLUS II uses the PSCRIPT keyword as the *<device driver>* to specify the Postscript printer driver.

```
[ports]
lpt1:=lp -c "%s"
lpt2:=lp -c -dps1700 "%s"
lpt3:=
...
```

The `[ports]` section lists the communication and printer ports available to MAX+PLUS II. The Windows `LPT`*n*`:` variables are equated to UNIX commands. In this example, LPT1 and LPT2 are equated to the print command **lp**. MAX+PLUS II prints its output to an intermediate Postscript file, which is then substituted for the term `"%s"`. The term `-dps1700` in the example refers to a UNIX printer named ps1700 that should be defined in the UNIX **printcap** file.

```
[PrinterPorts]
Apple LaserWriter II NT=PSCRIPT,LPT1:,15,90
Postscript Printer QMS=PSCRIPT,LPT2:,15,90
```

The `[PrinterPorts]` section lists the active and inactive output devices that can be accessed by the printer drivers, specifies the ports to which the output devices are connected, and specifies time-out values. In the example, the Apple LaserWriter II NT printer is connected to the `PSCRIPT` queue, and is connected to `LPT1`. MAX+PLUS II ignores the time-out values.

Printer Fonts

You can use the `[PSFontSubstitutes]` section of the **win.ini** file to specify aliases for actual printer fonts to match the fonts in MAX+PLUS II. The following example shows the default font substitute list:

```
[PSFontSubstitutes]
Helv=Helvetica
helvetica=Helvetica
MS Sans Serif=Helvetica
Tms Rmn=Times Roman
MS Serif=Times Roman
Times New Roman=Times Roman
Arial=Helvetica
courier=Courier
```

Glossary

This glossary defines selected terms used in MAX+PLUS II documentation.

☞ Choose **Glossary** (Help menu) to view the full MAX+PLUS II glossary on-line.

A

ACF *see* Assignment & Configuration File.

active-high node A node that is activated when it is assigned a value of one (`1` in AHDL and Verilog HDL or `'1'` in VHDL) or `VCC` (e.g., `ena`, `clk`).

active-low node A node that is activated when it is assigned a value of zero (`0` in AHDL and Verilog HDL or `'0'` in VHDL) or `GND` (e.g., `clrn`, `prn`, `oen`). In AHDL design files, an active-low node should be assigned a default value of `VCC` with the Defaults Statement.

ADF *see* Altera Design File.

AHDL *see* Altera Hardware Description Language.

Altera Design File (.adf) An ASCII-format file (with the extension **.adf**) for Boolean equation entry, used with Altera's A+PLUS software. ADFs use a netlist format and Boolean equations to describe a design. The MAX+PLUS II Compiler automatically translates an ADF into a Compiler Netlist File (**.cnf**) during project compilation.

An ADF is also generated when a State Machine File (**.smf**) is compiled.

Altera Hardware Description Language (AHDL) A high-level, modular language that is completely integrated into the MAX+PLUS II system. You can create AHDL Text Design Files (**.tdf**) with the

MAX+PLUS II Text Editor or any standard text editor, then compile, simulate, and program your projects within MAX+PLUS II. AHDL supports Boolean equation, state machine, conditional, and decode logic. AHDL also allows you to create and use parameterized functions, and includes full support for functions in the Library of Parameterized Modules (LPM).

Text Design Export files (**.tdx**) and Text Design Output Files (**.tdo**) generated by the MAX+PLUS II Compiler are also written in AHDL syntax.

Altera Megafunction Partner Program (AMPP) A program that offers support to third-party vendors to create and distribute megafunctions for use with MAX+PLUS II. You must enter a password in the **Megacore/AMPP Licenses** dialog box (accessed through the **Authorization Code** command on the Options menu) to enable a particular megafunction for implementation in a design file.

Additionally, some vendors may provide the option to view and edit a megafunction design file.

For information on current AMPP vendors, available megafunctions, and passwords, contact Altera Marketing.

ancillary file A file that is associated with a MAX+PLUS II project, but is not a design file in the project hierarchy tree. Most ancillary files also do not contain design logic. User-editable ancillary files with the same filename as the project appear in the Hierarchy Display window. See the following list:

Editable Ancillary Files:
Assignment & Configuration File (**.acf**)
Assignment & Configuration Output File (**.aco**)
Command File (**.cmd**)
EDIF Command File (**.edc**)
Fit File (**.fit**)
FLEX Chain File (**.fcf**)
Hexadecimal (Intel-format) File (**.hex**)
History File (**.hst**)
Include File (**.inc**)
Jam File (**.jam**)
JTAG Chain File (**.jcf**)
Library Mapping File (**.lmf**)
Log File (**.log**)
Memory Initialization File (**.mif**)
Memory Initialization Output File (**.mio**)
Message Text File (**.mtf**)
Programmer Log File (**.plf**)
Report File (**.rpt**)
Serial Vector Format File (**.svf**)
Simulator Channel File (**.scf**)
Standard Delay Format (SDF) Output File (**.sdo**)
Symbol File (**.sym**)
Table File (**.tbl**)
Tabular Text File (**.ttf**)
Text Design Export File (**.tdx**)
Text Design Output File (**.tdo**)
Timing Analyzer Output File (**.tao**)
Vector File (**.vec**)
VHDL Memory Model Output File (**.vmo**)

Non-Editable Ancillary Files:
Compiler Netlist File (**.cnf**)
Hierarchy Interconnect File (**.hif**)
JEDEC File (**.jed**)
Node Database File (**.ndb**)
Programmer Object File (**.pof**)
Raw Binary File (**.rbf**)
Serial Bitstream File (**.sbf**)
Simulator Initialization File (**.sif**)
Simulator Netlist File (**.snf**)
SRAM Object File (**.sof**)

area marquee In the Graphic or Symbol Editors, the rectangular boundary surrounding an area selection, which is created by dragging Button 1 with the Selection tool.

In the Hierarchy Display, the rectangular border that is visible as you drag the mouse to select an area. The marquee is visible only while you are dragging the mouse.

area selection A defined rectangular region that includes one or more adjacent objects. In the Graphic and Symbol Editors, this area is contained within a rectangular border called an area marquee. In the Waveform Editor, Floorplan Editor, and Hierarchy Display, all objects within an area selection are highlighted.

Area selection is the process of selecting multiple contiguous objects by dragging Button 1 with the Selection tool. In the Waveform Editor, such "objects" can consist of adjacent nodes and groups, whole waveforms, or intervals on one or more waveforms. In the Floorplan Editor, such "objects" can consist of adjacent pins, nodes, logic cells, or assignment bins. In the Hierarchy Display, file icons can be selected.

In the Graphic and Symbol Editors, symbols, arcs, circles, diagonal lines, and text blocks must lie completely within the area marquee to be selected. When an orthogonal line crosses the marquee, only the portion within the marquee is selected.

array *see* group.

ASCII American Standard Code for Information Interchange. Text editing software used for any MAX+PLUS II text file, e.g., Text Design File (**.tdf**), Library Mapping File (**.lmf**), or Vector File (**.vec**), must conform to this textual data coding system.

assignment In AHDL and VHDL, assignment refers to the transfer of a value to a symbolic name or group, usually through a Boolean equation. The value on the right side of the equation is assigned to the symbolic name or group on the left.

assignment (resource) *see* resource assignment.

Assignment & Configuration File (.acf) An ASCII file (with the extension **.acf**) that stores information about probe, pin, location, chip, clique, logic option, timing, connected pin, local routing, and device assignments, as well as configuration settings for the Compiler, Simulator, and Timing Analyzer for an entire project.

The ACF stores information entered with menu commands in all MAX+PLUS II applications, as well as pin, location, and chip assignments entered in the Floorplan Editor window. You can also edit an ACF manually in a Text Editor window.

B

back-annotation The process of copying device and resource assignments made by the Compiler, which are stored in the Fit File (**.fit**), into the Assignment & Configuration File (**.acf**) for a project. The back-annotation process preserves the current fit in future compilations.

background process An application or command that can run unattended as you perform another task and which can generate its own set of messages in a Message Processor window. The following

MAX+PLUS II applications and commands are background processes:

- Compiler
- Programmer
- Simulator
- Timing Analyzer
- ACF Reader
- Waveform Editor **Import Vector File** command (File menu)
- MAX+PLUS II **Project Archive** command (File menu)

balloon text Pop-up text in the Floorplan Editor that provides information on an item under the mouse pointer, such as a pin, I/O cell, logic cell, embedded cell, or an assignment bin. Information is displayed in the following formats:

<node name> **@** *<cell number>*

<pin name> **@** *<pin number>* **(***<pin function>* **(***<dedicated pin name>***))**

where the *<pin name>* or *<cell name>* is replaced by the text `<none>` if no item is assigned to a particular resource. If multiple functions are assigned to the resource, the first two names are listed, followed by the text `etc.` if there are additional names. In a last compilation floorplan, the text `(unrouted)` appears after the pin or node name for items that did not fit successfully.

batch mode The simulation mode in which Simulator commands are executed from a Command File (**.cmd**) rather than from on-screen options or menu commands.

binary The base 2 number system (radix). Binary digits are 0 and 1.

Boolean logic Logic that obeys the theorems of Boolean algebra (George Boole, "The Laws of Thought," 1854). The Boolean portion of a design is the portion which can be implemented in the `AND-OR` matrix of a device.

branches The extensions of the hierarchy tree that represent the different levels of the hierarchy. A branch consists of a design filename, a file icon, and any ancillary file icons. The intersections of branches are indicated by "+" and "-" branch buttons. Connection arrows lead from higher-level branches to lower-level branches.

breakpoint A user-defined set of conditions that will interrupt simulation when fulfilled.

buried node A combinatorial or registered signal that does not drive an output pin.

buried register A register in an Altera device that does not drive its output to a pin. A buried register can be located on an I/O pin or on a logic cell that has no output to a pin. A buried register can be used to implement internal logic.

bus A thick line in a Graphic Editor file that represents multiple nodes. A bus carries multiple signals between components of a design, and can represent from 2 to 256 nodes (i.e., bits).

In AHDL and Waveform Editor files, a group is synonymous with a bus.

In VHDL, a bus is a guarded signal that may have its drivers, i.e., signal sources, turned off. In VHDL, a bus is called an array, and is not limited to 256 symbolic names. An example of an array type is `STD_LOGIC_VECTOR`. See *Section 3.2.1:*

Array Types in the ***IEEE Standard VHDL Language Reference Manual*** for more information. Only one- and two-dimensional arrays of scalar elements are supported.

In Verilog HDL, a bus is an array of nets, and is limited to 256 symbolic names. See section *3.3: Vectors* in the ***IEEE Standard Hardware Description Language Based on the Verilog Hardware Description Language*** manual for more information.

bus (or group) name The name of a bus (or group) of up to 256 nodes.

A single-range or dual-range name consists of up to 32 name characters, followed by one or two ranges of numbers or arithmetic expressions in brackets. (Dual-range names are not supported in Waveform Editor files.) The start and end of the number range are separated by two periods. Each number in the sequence represents an individual node (or bit).

Example: bus `a[4..1]` consists of the nodes `a4`, `a3`, `a2`, and `a1`.

Example: bus `b[2..1][1..0]` consists of the nodes `b2_1`, `b2_0`, `b1_1`, and `b1_0`.

A sequential name, consisting of a comma-separated list of names, can be entered in AHDL Text Design Files (**.tdf**) and Graphic Design Files (**.gdf**). In TDFs only, this list of names must be enclosed in parentheses. Sequential bus names can include single- and dual-range bus names.

Example: `a[3..0],dout[6..4],z3`

The first name in the series of names in a single-range, dual-range, or sequential name is the most significant bit (MSB) of the bus; the last name is the least significant bit (LSB).

An arbitrary bus name, consisting of up to 32 name characters, can be entered in a Waveform Design File (**.wdf**), Simulator Channel File (**.scf**), or Vector File (**.vec**). An arbitrary bus name does not indicate how many members are included in the bus.

bus pinstub The location on the boundary of a mega- or macrofunction symbol, represented by an "x" in the Symbol File (**.sym**), that represents multiple inputs or outputs to the function. A bus (thick line) drawn in a Graphic Editor file must connect to a bus pinstub with the same number of bits to be recognized as a connection to the function.

ByteBlaster A Parallel download cable that allows PC users to program and configure devices in-system. The ByteBlaster provides programming support for MAX 7000S and MAX 9000 devices, and configuration support for FLEX 6000, FLEX 8000, and FLEX 10K devices. Multi-device JTAG chain programming and configuration are also available for FLEX 10K, MAX 7000S, and MAX 9000 devices. Multi-device FLEX chain configuration is available for FLEX 6000, FLEX 8000, and FLEX 10K devices.

The ByteBlaster is connected to a parallel printer port on a PC via a fully populated DB25-to-DB25 cable. The ByteBlaster's 10-pin female plug connects to a 10-pin male header on the circuit.

C

chip A group of logic functions defined as a single, named unit. A chip is assigned to an actual device by either the user or the Compiler.

You can make chip assignments on logic functions in design files. Items that are assigned to the same chip are placed in the same device during compilation. The term device always refers to an actual programmable logic device, whereas the term chip always refers to a group of logic functions.

When the Compiler processes a project, each chip name is assigned to a corresponding programming file for a particular device.

Classic An Altera device family based on Altera's original EPROM-based EPLD architecture. MAX+PLUS II provides support for the following Classic devices: EP600I, EP610, EP610I, EP900I, EP910, EP910I, EP1800I, and EP1810 devices.

Clear An input signal that resets a register. A synchronous Clear signal resets on each rising or falling Clock edge. An asynchronous Clear signal resets regardless of the Clock signal.

clique A group of logic functions defined as a single, named unit. The Compiler attempts to keep clique members together when it fits the project. A clique assignment allows you to group all logic on a speed-critical path, thus improving performance.

If possible, all clique members are assigned to the same LAB. If the clique members will not fit into a single LAB, they are placed in the same row (in FLEX 10K, FLEX 8000, FLEX 6000, and MAX 9000 devices only) or the same device.

Clock A signal that triggers registers.

In a flipflop or state machine, the Clock is an edge-sensitive signal. The output of the flipflop can change only on the Clock edge. For example, in a D flipflop, the input value is stored and placed on the output at the Clock edge.

In some cases, MAX+PLUS II lists the Latch Enable input to a latch as a Clock, e.g., in a Delay Matrix timing analysis.

Clock Enable The level-sensitive signal on an enabled flipflop, i.e., a flipflop with an "E" suffix, including DFFE, TFFE, SRFFE, and JKFFE. When the Clock Enable is low, Clock transitions on the Clock input to the flipflop are ignored.

column A vertical line of LABs connected by a column FastTrack Interconnect path in a FLEX 10K, FLEX 8000, FLEX 6000, or MAX 9000 device.

COM or **RS-232 port** An RS-232 serial communication port on a PC or UNIX workstation. The BitBlaster, which is used to configure and program devices in-system, must connect to a COM port.

combinatorial feedback Feedback from a logic cell that goes back into the device's logic array. It is the direct function of the inputs to a logic cell, and does not retain values from earlier inputs.

combinatorial output Output from a logic cell that is a direct function of the inputs, without regard to the Clock; i.e., it does not retain values resulting from earlier inputs.

Command File (.cmd) An ASCII text file (with the extension **.cmd**) that contains commands for batch-mode simulation.

comment In the Graphic and Symbol Editors, a comment is a free-floating block of text used to document the design. It is not associated with any object. A comment stands alone anywhere within Graphic Editor files. A comment also stands alone within the symbol border of a Symbol Editor file. Comments are ignored by the Compiler, and can be used to document various sections of a file.

In the Waveform Editor, a comment is a line of text used to annotate the waveforms in the waveform drawing area . It is not associated with any waveform. A comment is anchored to the time on the time scale where the first character is entered. A label appears in the Name field to indicate a comment line; when a comment is added between two existing nodes, it appears in a blank space, which is inserted between the waveforms. Comments are ignored by the Compiler.

In all MAX+PLUS II text files except VHDL Design Files (**.vhd**), Verilog Design Files (**.v**), and Assignment & Configuration Files (**.acf**), e.g., in Report Files (**.rpt**), Vector Files (**.vec**), and Text Design Files (**.tdf**), a comment is any string of characters enclosed in percent symbols (%). You can insert comments wherever white space is allowed in text files.

In VHDL Design Files and ACFs, comments begin with two dashes (- -) and continue to the End-of-Line. AHDL TDFs also support VHDL-style comments. If you use a VHDL-style comment in a TDF, you must separate the two dashes from any preceding symbolic name with at least one space.

In Verilog Design Files, comments begin with two slashes (//) and continue to the End-of-Line. Verilog Design Files and ACFs also support comments consisting of any string of characters enclosed between /* and */ characters.

Compiler Netlist File (.cnf) A binary file (with the extension **.cnf**) that contains the data from a design file. The CNF is created by the Compiler Netlist Extractor module of the MAX+PLUS II Compiler.

Configuration EPROM Altera's family of serial EPROMs, which are designed to configure FLEX 6000, FLEX 8000, and FLEX 10K devices. This device family includes the EPC1, EPC1213, EPC1064, EPC1064V, and EPC1441 devices.

connection dot A dot entered at an intersection of two signal lines (nodes or buses) in a Graphic Editor file. The connection dot indicates that the signals are logically connected.

construct A unit in a text design language such as AHDL, VHDL, Verilog HDL, or EDIF.

continuity checking A test for open circuits between device pins and programming adapter sockets. This test verifies that a device is properly seated in the socket of the adapter.

cutoff node A node that is excluded from timing analysis. The signal associated with a node can be cut off from a timing analysis by tagging it with the **Timing Analysis Cutoff** command.

D

database A flattened representation of all design files in a MAX+PLUS II project hierarchy. The database is used internally by Compiler modules during compilation.

decimal The base 10 number system (radix). Decimal digits are 0 through 9.

In AHDL,VHDL, and Verilog HDL no special notation is needed to indicate decimal digits.

default Simulator Channel File (.scf) A Simulator Channel File (**.scf**) that can contain all nodes and groups that are in the Simulator Netlist File (**.snf**) for a project. It is created automatically with the **Enter Nodes from SNF** command (Node menu) in the Waveform Editor.

default timing tagging The Timing Analyzer provides the following default node tagging for timing analysis:

Analysis Mode:	Default Tagging:
Delay Matrix	All input pins are sources; all output pins are destinations.
Setup/Hold Matrix	All input pins are sources; all data and Clock inputs to registers, Latch Enable inputs to latches, and data, address, and Write Enable inputs to asynchronous RAM are destinations.
Registered Performance	All Q outputs of registers are sources; all data and Clock Enable inputs to registers are destinations.

delimiter A text string, character, or keyword used to define the beginning or the end of a statement or construct in a text file.

For example, [and] are delimiters of AHDL group ranges and % is a comment delimiter in many MAX+PLUS II text files.

design file A file that contains logic for a MAX+PLUS II project and is compiled by the Compiler. The following files are design files:

- Altera Design File (**.adf**)
- EDIF Input File (**.edf**) *
- Graphic Design File (**.gdf**) *
- OrCAD Schematic File (**.sch**) *
- State Machine File (**.smf**)
- Text Design File (**.tdf**) *
- Verilog Design File (**.v**)
- VHDL Design File (**.vhd**) *
- Waveform Design File (**.wdf**)
- Xilinx Netlist Format File (**.xnf**)

An asterisk (*) indicates the design files that can exist as top-level files in hierarchical projects. Other design files must be the only design file in a project or must exist at the bottom level of a hierarchical project.

destination node A node that is tagged (designated) as the destination of a signal for the purpose of timing analysis. A destination node is tagged with the **Timing Analysis Destination** command (Utilities menu), and can be any node that is the input to a logic function or a pin.

device A device refers to an Altera programmable logic device, including Classic, MAX 5000, MAX 7000, MAX 9000,

FLEX 6000, FLEX 8000, and FLEX 10K device families.

Altera also offers Configuration EPROM devices which are used to configure FLEX 6000, FLEX 8000, and FLEX 10K devices.

device assignment A device assignment assigns a user-specified block of logic functions, called a chip, to a specific Altera device.

device family A group of Altera programmable logic devices with the same fundamental architecture. Altera families include the Classic, MAX 5000, MAX 7000, MAX 9000, FLEX 6000, FLEX 8000, and FLEX 10K, device families.

device option An option that controls a device. Altera devices offer the following device options:

Option:	Device Family:
Auto-Restart Configuration on Frame Error	FLEX 6000, FLEX 8000, and FLEX 10K
Disable Start-Up Time-Out	FLEX 8000
Enable Chip-Wide Output Enable	FLEX 6000 and FLEX 10K
Enable Chip-Wide Reset	FLEX 6000 and FLEX 10K
Enable DCLK Output in User Mode	FLEX 8000
Enable INIT_DONE Output	FLEX 6000 and FLEX 10K
Enable JTAG Support	MAX 7000S, FLEX 6000, and FLEX 8000
Enable LOCK Output	FLEX 10K
JTAG User Code	FLEX 10K
Low-Voltage I/O	All
Release Clears Before Tri-States	FLEX 6000, FLEX 8000, and FLEX 10K
Security Bit	Classic, MAX 5000, MAX 7000, and MAX 9000
Turbo Bit	Classic, MAX 5000, MAX 7000, and MAX 9000 (*Logic Cell Turbo Bit* can be applied to all logic cells in a MAX 7000 or MAX 9000 device.)
Use Low-Voltage Configuration EPROM	FLEX 6000 and FLEX 10K
User Code	MAX 7000S and MAX 9000
User-Supplied Start-Up Clock	FLEX 6000, FLEX 8000, and FLEX 10K

dual I/O feedback A combination of pin feedback and register or combinatorial feedback on the same logic cell.

dynamic models Models that represent actual combinatorial logic in timing Simulator Netlist Files (**.snf**).

Dynamic models are generated for the logic in a timing SNF when the Compiler's **Optimize Timing SNF** command (Processing menu) is turned on. Instead of processing the combinatorial logic, the Simulator or Timing Analyzer refers to the representative dynamic model.

Dynamic models allow a simulation to run faster; however, the Compiler requires additional time to generate the SNF.

E

EAB *see* Embedded Array Block.

EC *see* embedded cell.

EDIF Electronic Design Interchange Format. An industry-standard format for the transmission of design data.

You can generate an EDIF 2 0 0 or 3 0 0 netlist file from a schematic design or from a VHDL or Verilog HDL design that has been processed with an appropriate industry-standard synthesis tool and then import the file into MAX+PLUS II as an EDIF Input File (**.edf**). MAX+PLUS II supports EDIF Input Files that contain functions from the Library of Parameterized Modules (LPM). The MAX+PLUS II Compiler can also generate one or more EDIF Output Files (**.edo**) in either EDIF 2 0 0 or 3 0 0 format that contain functional or timing information for simulation with a standard EDIF simulator.

The MAX+PLUS II Compiler's EDIF Netlist Reader and EDIF Netlist Writer modules have been awarded the Electronic Industries Association's (EIA) EDIF version 3 0 0 Self-Verification Seal of Approval. This award indicates that MAX+PLUS II EDIF 3 0 0 support has successfully completed the testing process to ensure compliance with the EDIF 3 0 0 Netlist View standard.

EDIF Command File (.edc) An ASCII text file (with the extension **.edc**) used to customize the format of EDIF Output Files (**.edo**) created by the Compiler's EDIF Netlist Writer module.

EDIF Input File (.edf) An EDIF version 2 0 0 or 3 0 0 netlist file generated by any standard EDIF netlist writer. EDIF Input Files (with the extension **.edf**) can be compiled by the MAX+PLUS II Compiler. MAX+PLUS II supports EDIF Input Files that contain functions from the Library of Parameterized Modules (LPM).

EDIF Output File (.edo) An EDIF version 2 0 0 or 3 0 0 netlist file (with the extension **.edo**) generated by the EDIF Netlist Writer module of the Compiler. This file can be exported to an industry-standard UNIX workstation or PC environment for simulation.

EEPROM Electrically Erasable Programmable Read-Only Memory. A form of reprogrammable semiconductor memory in which the contents (program) can be erased by subjecting the device to appropriate electrical signals.

Embedded Array Block (EAB) A physically grouped set of 8 embedded cells that implement memory (RAM or ROM) or combinatorial logic in a FLEX 10K device. An EAB consists of an embedded cell array, with data, address, and control signal inputs and data outputs that are optionally registered.

A single EAB can implement a memory block of 256 × 8, 512 × 4, 1,024 × 2, or 2,048 × 1 bits. Each embedded cell within the EAB implements up to 256 bits of memory. For memory blocks of these sizes, an EAB has 8, 4, 2, or 1 outputs, respectively. Multiple EABs can be combined to create larger memory blocks.

The EAB is fed by row interconnect paths and a dedicated input bus.

embedded cell (EC) A memory element that exists in the embedded array of a FLEX 10K device, and which can implement memory (RAM or ROM) or combinatorial logic. An Embedded Array Block (EAB) consists of a group of 8 embedded cells that can implement a memory block of 256 × 8, 512 × 4, 1,024 × 2, or 2,048 × 1 bits. Each embedded cell within an EAB implements up to 256 bits of memory. Depending on the depth of the memory, up to 8 of the embedded cells in an EAB have outputs. For memory blocks of 256 × 8, 512 × 4, 1,024 × 2, or 2,048 × 1 bits, an EAB has 8, 4, 2, or 1 outputs, respectively.

Embedded cells have "numbers" of the format EC<*number*>_<*row letter*>, where <*number*> ranges from 1 to 8 and <*row letter*> consists of the row letter of the EAB.

EPLD Erasable Programmable Logic Device, i.e., an Altera device that is a member of the Classic, MAX 5000, MAX 7000, or MAX 9000 device families.

EPROM Erasable Programmable Read-Only Memory. A form of reprogrammable semiconductor memory in which the contents (program) can be erased by subjecting the device to ultraviolet light of the proper wavelength.

evaluated function An mathematical function that evaluates an arithmetic expression and returns a value based on one or more arguments. The AHDL Define Statement can be used to create evaluated functions. The following example shows the definition of the evaluated function MAX:

```
DEFINE MAX(a,b) = (a > b) ? a : b;
```

expander product term A single product term with an inverted output that feeds back into the Logic Array Block (LAB) of a MAX 5000, MAX 7000, or MAX 9000 device.

An uncommitted expander product term that can be shared with other logic cells in the same LAB is called a shareable expander; a product term that has been shared in this manner is called a shared expander.

In MAX 7000 and MAX 9000 devices only, an expander product term that is "borrowed" from an adjacent logic cell in the same LAB is called a parallel expander.

extension *see* filename extension.

F

family-specific mega- or macrofunction An Altera-provided mega- or macrofunction that contains logic optimized for the architecture of a specific device family.

The functionality of a family-specific mega- or macrofunction is always the same, regardless of the device family for which it is designed. However, the actual primitives and nodes used within the mega- or macrofunction file can vary from family to family to take advantage of different device architectures, thus providing higher performance and/or more efficient implementation.

fan-in and **fan-out** Fan-in refers to input signals that feed the input equations of a logic cell.

Fan-out refers to output signals that are fed by the output equations of a logic cell.

FastTrack Interconnect Dedicated connection paths that span the entire width and height of a FLEX 6000, FLEX 8000, MAX 9000, or FLEX 10K device. These connection paths allow signals to travel between all Logic Array Blocks (LABs) in a device.

FCF *see* FLEX Chain File.

file icon An icon that appears in a MAX+PLUS II application window and represents a file in the current hierarchy tree. Double-clicking Button 1 on an icon opens the file that it represents.

In the Hierarchy Display, the file icon shows which MAX+PLUS II editor can open the file. The filename extension is displayed at the bottom of the file icon to show the file type; the filename is displayed to the left of the file icon.

In the Compiler, the file icons show input and output files for the current project.

filename The name of a design file, ancillary file, or other file, without the extension.

A single filename can contain up to 32 name characters, plus a 3-character filename extension. A full pathname plus filename and extension can contain up to 128 characters.

Because Windows 3.1 and Windows for Workgroups 3.11 support only 8-character filenames, MAX+PLUS II maps longer filenames on all Windows operating systems to 8-character filenames by default. These filename mappings are stored in the **maxplus2.idx** file in each directory that contains long filenames. However, you can override this behavior and us the built-in support for long filenames available in Windows NT and Windows 95 by setting the `USE_WINNT_LONG_FILENAMES` variable in the `[system]` section of your **maxplus2.ini** file to `ON`.

In the Hierarchy Display window, a filename, along with the file icon and filename extension, represents a file in the current hierarchy tree.

filename extension The one, two, or three-letter extension of a filename that follows a period (`.`).

In the Hierarchy Display window, a filename extension, along with the filename and the file icon, represents a file in the current hierarchy or the current project.

Fit File (.fit) An ASCII file (with the extension **.fit**) generated by the Compiler that documents pin, logic cell, I/O cell, embedded cell, chip, and device assignments made during the last compilation. Assignments are recorded in Assignment & Configuration File (**.acf**) syntax.

The Fit File can be used for back-annotation and for functional testing in the Simulator and Programmer. To preserve assignments permanently, Fit File assignments can be back-annotated into a project's ACF with the **Back-Annotate Project** command (Assign menu).

You can also display a read-only version of Fit File information from the most recent project compilation in the Floorplan Editor.

FLEX Chain File (.fcf) An ASCII file (with the extension **.fcf**) that stores programming file names for use in configuring multiple FLEX 6000, FLEX 8000, or FLEX 10K devices in a Passive Serial configuration scheme. An FCF saves the information entered with the Programmer's **Multi-Device FLEX Chain Setup** command (FLEX menu).

FLEX 6000 An Altera device family based on Flexible Logic Element MatriX architecture. This SRAM-based family offers high-performance, register-intensive, high-gate-count devices. The FLEX 6000 device family includes the EPF6016 device.

FLEX 8000 An Altera device family based on Flexible Logic Element MatriX architecture. This SRAM-based family offers high-performance, register-intensive, high-gate-count devices. The FLEX 8000 device family includes the EPF8282V, EPF8282A, EPF8282AV, EPF8452A, EPF8636A, EPF8820A, EPF81188A, and EPF81500A devices.

Altera recommends using FLEX 8000A devices rather than FLEX 8000 devices for all new designs.

FLEX 10K An Altera device family based on Flexible Logic Element MatriX architecture. This SRAM-based family offers high-performance, register-intensive, high-gate-count devices with embedded arrays. The FLEX 10K device family includes the EPF10K100, EPF10K70, EPF10K50, EPF10K40, EPF10K30, EPF10K20, and EPF10K10 devices.

FLEX 10K devices, which include EPF10K50V, EPF10K130V, and EPF10K250A devices, are enhanced versions of FLEX 10K devices, and are function-, pin-, and programming-file-compatible with FLEX 10K devices. FLEX 10KA devices differ from FLEX 10K devices in that they are 3.3-V versions of FLEX 10K devices.

The EPF10K100GC503-3DX device includes built-in ClockLock and ClockBoost phase-locked loop circuitry.

flipflop or **register** An edge-triggered, clocked storage unit that stores a single bit of data. A low-to-high transition on the Clock signal changes the output of the flipflop, based on the value of the data input(s). This value is maintained until the next low-to-high transition of the Clock, or until the flipflop is preset or cleared.

Depending on the architecture of the device family, a register can be programmed as a level-sensitive flow-through latch or as an edge-triggered D,T, JK, or SR flipflop.

In Verilog HDL, "register" is also used to describe the abstraction of a data storage device that the MAX+PLUS II Compiler uses to infer registers.

f_{MAX} (maximum Clock frequency) The maximum Clock frequency that can be achieved without violating internal setup and hold time requirements.

f_{MAX} is also a timing assignment that specifies the minimum acceptable Clock frequency. In MAX+PLUS II, you can

specify a required $\mathbf{f_{MAX}}$ for an entire project and/or for any input pin (`INPUT` or `INPUTC`), bidirectional pin (`BIDIR` or `BIDIRC` input function), or a register.

Function Prototype Specifies the ports (pinstubs) of a primitive, megafunction, or macrofunction in AHDL. A Function Prototype consists of the name of the function, and a list of its inputs and outputs. For mega- and macrofunctions, the Function Prototype can also contain parameters that are used to specify the characteristics of the function. Function Prototypes are specified in the Function Prototype Statement. They are often stored in Include Files (**.inc**). Include Files that contain Function Prototypes for Altera-provided mega- and macrofunctions are located in the **\maxplus2\max2lib\mega_lpm** and **\maxplus2\max2inc** directories created during installation, respectively. (On a UNIX workstation, the **maxplus2** directory is a subdirectory of the **/usr** directory.)

To implement an instance of a mega- or macrofunction in AHDL, its logic must be defined in a design file and its Function Prototype must be declared. (Function Prototypes are optional for primitives.) You can then create an instance of the function with an Instance Declaration or an in-line reference.

When you use a Module Instantiation in Verilog HDL, the MAX+PLUS II Compiler uses the port name and ordering information in AHDL Include Files that contain Function Prototypes to implement an instance of the logic function.

G

GDF *see* Graphic Design File.

glitch or **spike** A signal value pulse that occurs when a logic level changes two or more times over a short period.

When the Simulator is in timing or linked simulation mode, you can define the length of a glitch and monitor the project for pulses shorter than the defined value. Glitch detection is not available in functional simulation mode.

global signal A pin- or logic-driven signal that passes through the global routing on a device before performing its specified function. Clock, Preset, Clear, and Output Enable signals can be global signals.

Logic-driven global signals are available only in FLEX 6000 devices.

A global signal can be set in various ways:

- During design entry with a `GLOBAL` primitive. You can use a dedicated input pin to drive a global signal directly by feeding its output directly to a `GLOBAL` primitive. You can also use the output of a logic function as a global signal by feeding its output directly to a `GLOBAL` primitive. A logic-driven global signal consumes a dedicated global input pin.
- With the *Automatic Global* option in the **Global Project Logic Synthesis** dialog box (Assign menu). The Compiler chooses the pin-driven signal that feeds the most flipflops as a global Clock, Preset, or Clear, and the signal that feeds the most `TRI` buffers is chosen as the global Output Enable.

- With the *Global Signal* logic option in the **Individual Logic Options** dialog box, which you can open from the **Logic Options** dialog box (Assign menu). When this option is turned on for an input pin or for a single-output logic function, it is equivalent to using a `GLOBAL` primitive. Turning this logic option off prevents an input pin from being used as a global signal.

GND A low-level input voltage.

`GND` is the default inactive node value. In an AHDL Text Design File (**.tdf**), `GND` is used as a predefined constant and keyword. In a VHDL Design File (**.vhd**), `GND` is represented by `'0'`. In a Verilog Design File (**.v**), `GND` is represented by `0`. In a Graphic Editor file, `GND` is a primitive symbol. `GND` is represented as a low (0) logic level in the Simulator and Waveform Editor.

Graphic Design File (.gdf) A schematic design file (with the extension **.gdf**) created with the MAX+PLUS II Graphic Editor.

An OrCAD Schematic File (**.sch**) is automatically translated into a GDF and treated as a GDF in the MAX+PLUS II Graphic Editor and Compiler.

Gray code A counting scheme in which only one bit at a time changes value between consecutive count values. In contrast, a binary count sequence does not preclude more than one bit changing at consecutive count values. When only one bit changes, noise susceptibility is reduced in the circuit.

group or **array** In AHDL, a group is a collection of up to 256 symbolic names that are treated as a unit. A group name can be specified with a single-range group name, dual-range group name, or sequential group name format.

In VHDL, a group is called an array, and is not limited to 256 symbolic names. Examples of array types are `STD_LOGIC_VECTOR` and `BIT_VECTOR`. See *Section 3.2.1: Array Types* in the ***IEEE Standard VHDL Language Reference Manual*** for more information. Only one- and two- dimensional arrays of scalar elements are supported.

In Verilog HDL, a group is called an array, and is limited to 256 symbolic names. Examples of array types are memories (which are arrays of register elements or words) and arrays of gate instances and registers. The elements, instances, or registers in the array are specified with a range. See *Section 3.3: Vectors*, *Section 3.8: Memories*, and *Section 7: Gate and Switch Level Modeling* in the ***IEEE Standard Hardware Description Language Based on the Verilog Hardware Description Language*** manual for more information.

In the Waveform Editor and Simulator, a group is a collection of up to 256 nodes that are treated as a unit. In these applications, a group name can be specified with an arbitrary group name or single-range group name format.

group name *see* bus name.

H

hard logic function A logic function in a design file that is not removed during standard logic synthesis and therefore can be assigned to a physical resource such as a specific device, pin, logic cell, or I/O cell.

In Graphic Design Files (**.gdf**) and Text Design Files (**.tdf**), hard logic primitives/ports include `INPUT`, `INPUTC`, `OUTPUT`, `OUTPUTC`, `BIDIR`, `BIDIRC`, `LCELL`, `MCELL`, `DFF`, `DFFE`, `TFF`, `TFFE`, `JKFF`, `JKFFE`, `SRFF`, `SRFFE`, and `LATCH`. However, `INPUT` and `INPUTC` primitives that do not affect project outputs are not considered to be hard logic functions. When `SOFT`, `TRI`, and `OPNDRN` primitives are not removed during logic synthesis, they are also hard logic primitives. A megafunction or macrofunction that contains a hard logic primitive is considered to be a hard logic function.

In Waveform Design Files (**.wdf**), hard logic functions are input nodes and output and buried nodes with registered and combinatorial node types.

hexadecimal The base 16 number system (radix). Hexadecimal digits are 0 through 9 and A through F.

Hexadecimal numbers are indicated with the following notation:

Language	Notation
AHDL	**X"**<*series of digits 0 to 9, A to F*>**"** *or* **H"**<*series of digits 0 to 9, A to F*>**"**
VHDL	**16#**<*series of digits 0 to 9, A to F*>**#**
Verilog HDL	**'h**<*series of digits 0 to 9, A to F*>

Examples:

`H"123AECF"` (AHDL)
`16#FF#` (VHDL)
`'h837FF` (Verilog HDL)

Hexadecimal (Intel-Format) File (.hex) An ASCII text file (with the extension **.hex**) in the Intel hexadecimal format.

The MAX+PLUS II Compiler and Simulator can use Hex Files as inputs to specify the initial contents of a memory (e.g., a ROM).

The MAX+PLUS II Compiler automatically creates output Hex Files containing configuration data for the Active Parallel Up (APU) configuration scheme for FLEX 8000 devices, and the Passive Serial (PS) configuration scheme for FLEX 6000 and FLEX 10K devices.

After compilation, you can also create Hex Files that support other configuration schemes for FLEX 6000, FLEX 8000, and FLEX 10K devices.

If your project uses memory and you use a Hex File to specify its initial contents, you should name the file with a name that is not the same as the project name or any chip name within the project. Because the Compiler automatically generates Hex Files as outputs for FLEX 6000, FLEX 8000, and FLEX 10K devices, these output files may overwrite your initial memory content files.

hierarchical node or symbol name The unique name for a node or symbol that is based on its location in the hierarchy of design files and the net ID number or the AHDL,VHDL, or Verilog HDL instance name of the logic function to which it is connected.

Every node and symbol in a project has a hierarchical name; you can also assign a node name or a probe name to a node.

Hierarchy Interconnect File (.hif) An ASCII file (with the extension **.hif**) created by the Compiler's Netlist Extractor module. This file specifies the hierarchical interconnections between design files in a project.

History File (.hst) An ASCII file (with the extension **.hst**) created by the MAX+PLUS II Simulator. This time period records all commands, buttons, and on-screen options that are used during a simulation session, as well as their output.

hold time On a flipflop, the hold time is the minimum time period for which a signal must be retained on an input pin that feeds the data input or Clock Enable after an active transition at the input pin that feeds the flipflop's Clock input.

On a latch, the hold time is the minimum time period for which a signal must be retained on an input pin that feeds the D input after an active transition at the input pin that feeds the Latch Enable input.

On an asynchronous RAM block, the hold time is the minimum time period for which a signal must be retained on an input pin that feeds the data or address inputs after an active transition at the input pin that feeds the RAM block's Write Enable input.

Internal hold times for flipflops, latches, and asynchronous RAM, which are not user-controllable, similarly constrain internally generated signals.

I

I/O cell An I/O cell is a register (also known as an I/O element) that exists on the periphery of a FLEX 10K, FLEX 8000, or MAX 9000 device. I/O cells permit short setup time.

☞ In pre-version 5.0 releases of MAX+PLUS II, I/O cells were known as peripheral registers.

I/O feedback Feedback from the output pin on an Altera device. It allows an output pin to be also used as an input pin.

I/O type The direction of signal travel for a node, pin, or state machine.

In the Graphic and Symbol Editors, pins and pinstubs can have I/O types of input, output, or bidirectional.

In AHDL, the I/O type of a port can be input, output, buried (i.e., buried output), machine input, or machine output.

In the Waveform Editor, the I/O type of a node can be input, output, or buried (i.e., buried output). Input and output I/O types can represent actual pin outputs; a buried I/O type always represents logic that does not feed a pin.

ICR *see* in-circuit reconfigurability.

in-circuit reconfigurability (ICR) The capability of SRAM-based devices, such as Altera's FLEX 6000, FLEX 8000, and FLEX 10K devices, to load configuration data at system power-up or during normal system operation after they have been mounted on a printed circuit board.

In-circuit reconfiguration can be performed an unlimited number of times with data from a local PROM such as an Altera Configuration EPROM, or with data downloaded by an external controller such as a CPU or the MAX+PLUS II

Programmer. The Programmer also provides the capability to configure one or more FLEX 10K devices in a JTAG chain and one or more FLEX 6000, FLEX 8000, or FLEX 10K devices in a FLEX chain.

in-system programmability (ISP) The capability of EEPROM-based devices, such as Altera's MAX 9000 and MAX 7000S devices, to be programmed after they have been mounted on a printed circuit board.

The MAX+PLUS II Programmer supports in-system programming via the BitBlaster serial download cable and the ByteBlaster parallel download cable. The Programmer also provides the capability to program multiple devices in a JTAG chain.

Include File (.inc) An ASCII text file (with the extension **.inc**) that can be imported into a Text Design File (**.tdf**) by an AHDL Include Statement. The Include File replaces the Include Statement that calls it. Include Files can contain Function Prototype, Define, Parameters, or Constant Statements. Include Files that contain Function Prototypes for Altera-provided mega- and macrofunctions are located in the **\maxplus2\max2lib\mega_lpm** and **\maxplus2\max2inc** directories created during installation, respectively. (On a UNIX workstation, the **maxplus2** directory is a subdirectory of the **/usr** directory.)

☞ When you use a Module Instantiation in Verilog HDL, the MAX+PLUS II Compiler uses the port name and ordering information in AHDL Include Files that contain Function Prototypes to implement an instance of the logic function.

insertion point The location at which text or graphics are inserted.

In a dialog box or in the Text Editor window, the insertion point appears as a flashing vertical bar. In the Graphic or Symbol Editor, it appears as a flashing square. In the Waveform Editor, an insertion point in the waveform drawing area appears as a short horizontal line that extends to the right of the Time cursor. In the node/group information area, a name or blank space that is selected is interpreted as an insertion point.

When you type text, it appears to the left of the insertion point, which moves to the right as you type. When you enter or paste symbols or waveforms, the upper left corner of the item(s) appears at the insertion point.

instance The use of a logic function in a design file. In the Graphic Editor, the instance is represented by the symbol (net) ID number in the lower left corner; in the Waveform Editor, it is the name of the node. In AHDL, instances are declared in one of two forms: an Instance Declaration that declares a variable of the type *<primitive>*, *<megafunction>*, or *<macrofunction>*, or an in-line logic function reference. In VHDL, instances of logic functions are declared with a Component Instantiation Statement; registers can also be implemented with Register Inferences. In Verilog HDL, instances are declared with Module Instantiations and Gate Instantiations.

In the Hierarchy Display, an instance of a mega- or macrofunction is represented by the function name, followed by a colon (:) and a net ID number. In an AHDL Variable Declaration and a VHDL Component Instantiation Statement, an instance is represented by the instance name followed by a colon and the function name. In a

Verilog HDL Module or Gate Instantiation, an instance is represented by the module or gate name, followed by the instance name.

interactive mode The simulation mode in which you choose on-screen options and buttons, and execute menu commands, with the keyboard or mouse.

ISP *see* in-system programmability.

J

Jam File (.jam) An ASCII file (with the extension **.jam**) in the Jam device programming and test language that stores programming data for programming, verifying, and blank-checking one or more in-system programmable devices in a JTAG chain. Jam files are use in embedded processor-type programming environments. Altera's MAX 7000S and MAX 9000 devices can be programmed with Jam files. The JTAG chain can contain any other device that complies with the IEEE 1149.1 JTAG specification, including FLEX 10K, FLEX 6000, and some FLEX 8000 devices.

You can generate Jam files with the **Create Jam or SVF File** command (File menu) in the Programmer or the Compiler.

JCF *see* JTAG Chain File.

JEDEC File (.jed) An ASCII file (with the extension **.jed**) that contains programming information. JEDEC Files provide an industry-standard format for transferring information between a data preparation system and a logic device programmer. The MAX+PLUS II Compiler automatically generates JEDEC Files for all Classic devices and the EPM5032 device during compilation.

The MAX+PLUS II Programmer can use a JEDEC File created with MAX+PLUS II, MAX+PLUS (DOS), A+PLUS, or PLDshell Plus to program the Altera devices listed above. The Programmer can also optionally save programming data plus functional test vectors in JEDEC File format.

JTAG boundary-scan testing Testing that isolates a device's internal circuitry from its I/O circuitry. This testing is made possible by the Joint Test Action Group (JTAG) Boundary-Scan Test (BST) architecture that is available in all FLEX 10K devices; all FLEX 8000 devices except the EPF8452A and EPF81188A; all FLEX 6000 devices; all MAX 9000 devices; and all MAX 7000S devices except the EPM7064S. Serial data is shifted into boundary-scan cells in the device; observed data is shifted out and externally compared to expected results. Boundary-scan testing offers efficient PC board testing, providing an electronic substitute for the traditional "bed of nails" test fixture.

The full or partial JTAG BST architecture in all FLEX 10K, MAX 9000, and MAX 7000S devices also supports in-system multi-device JTAG chain device programming and configuration.

JTAG chain *see* multi-device JTAG chain.

JTAG Chain File (.jcf) An ASCII file (with the extension **.jcf**) that stores device name, device order, and optional programming file name information for use in programming or configuring one or more devices in a JTAG chain. A JCF saves information entered with the Compiler or Programmer's **Create Jam or SVF File**

command (File menu) or **Multi-Device JTAG Chain Setup** command (JTAG menu).

K

keyword Words that are reserved for implementing syntax in files used as inputs to MAX+PLUS II, including AHDL Text Design Files (**.tdf**), Assignment & Configuration Files (**.acf**), Command Files (**.cmd**), EDIF Command Files (**.edc**), Library Mapping Files (**.lmf**), VHDL Design Files (**.vhd**), Verilog Design Files (**.v**) and Vector Files (**.vec**). For example, the keyword `OF` cannot be used as an unquoted symbolic name in an AHDL file.

L

LAB *see* Logic Array Block.

latch A level-sensitive clocked storage unit that stores a single bit of data. A high-to-low transition on the Latch Enable signal fixes the contents of the latch at the value of the data input until the next low-to-high transition of the Latch Enable.

Latch Enable A level-sensitive signal that controls a latch. When it is high, the input flows through the output; when it is low, the output holds its last value.

LC *see* logic cell.

least significant bit (LSB) The bit of a binary number that contributes the smallest quantity to the value of that number, i.e., the last member in a bus or group name. For example, the LSB for a bus or group named `a[31..0]` is `a[0]` (or `a0`).

Library Mapping File (.lmf) An ASCII text file (with the extension **.lmf**) used to map cells in EDIF Input Files (**.edf**) or symbols in OrCAD Schematic Files (**.sch**) to corresponding MAX+PLUS II primitives, megafunctions, and macrofunctions.

Library of Parameterized Modules (LPM) A technology-independent library of logic functions that are parameterized to achieve scalability and adaptability. Altera has implemented parameterized modules (also called "parameterized functions") from LPM version 2.1.0 that offer architecture-independent design entry for all MAX+PLUS II-supported devices. The MAX+PLUS II Compiler includes built-in compilation support for LPM functions used in schematic, AHDL, VHDL, Verilog HDL, and EDIF input files.

LMF *see* Library Mapping File.

Load An input signal that loads data into a register. A synchronous Load signal loads data on each rising or falling Clock edge. An asynchronous Load signal loads data regardless of the Clock signal.

local routing A resource assignment available for FLEX 6000 devices that assigns a fan-out of a node to be placed in logic cell in the same LAB as the node or in an adjacent LAB to the node. Local routing is also available between a node that is placed in a logic cell in an LAB on the periphery of a device and the output pin that it feeds. Local routing assignments ensure that the signals are connected with shared local interconnect, which is the fastest interconnect available. Therefore, you can maximize your project's performance by connecting logic on a speed-critical path with local routing.

location A generic term that refers to an assignable physical resource in the interior of an Altera device.

You can assign a logic function to one of the following locations:

- An individual logic cell
- An individual I/O cell
- An individual embedded cell
- A logic array block (LAB), embedded array block (EAB), row, or column

When you assign a logic function to a general location such as a LAB, EAB, row, or column, the Compiler can choose the best logic cell or embedded cell within the LAB, row, or column to use to implement the logic.

Log File (.log) An ASCII text file (with the extension **.log**) created by the MAX+PLUS II Simulator. The Log File records all commands, buttons, and on-screen options that are used during an interactive simulation session.

logic function or Design Entity A primitive, megafunction, macrofunction, or state machine, which may be represented as either a name or a symbol in a design file.

Logic Array Block (LAB) A physically grouped set of logic resources in an Altera device. An LAB consists of a logic cell array and, in some device families, an expander product term array. Any signal that is available to any one logic cell in the LAB is available to the entire LAB.

In Classic devices, the logic in the LAB shares a global Clock signal. The LAB is fed by a global bus and a dedicated input bus. (In an EP1810 device, an LAB is synonymous with a quadrant.) In MAX 5000 and MAX 7000 devices, the LAB is fed by a Programmable Interconnect Array (PIA) and a dedicated input bus. In FLEX 6000, FLEX 8000, MAX 9000, and FLEX 10K devices, the LAB is fed by row FastTrack Interconnect paths and a dedicated input bus.

logic cell (LC) The generic term for a basic building block of an Altera device. In Classic, MAX 5000, MAX 7000, and MAX 9000 devices, a logic cell (also called a macrocell) consists of two parts: combinatorial logic and a configurable register. The combinatorial logic allows a wide variety of logic functions. In FLEX 6000, FLEX 8000, and FLEX 10K devices, a logic cell (also called a logic element) consists of a look-up table (LUT), i.e., a function generator that quickly computes any function of four variables, and a programmable register to support sequential functions.

The register can be programmed as a flow-through latch; as a D, T, JK, or SR flipflop; or bypassed entirely for pure combinatorial logic. The register can feed other logic cells or feed back to the logic cell itself. Some logic cells feed output or bidirectional I/O pins on the device.

You can assign a logic function to a specific logic cell. You can also assign a logic function to a logic array block (LAB), a row, or a column to ensure that the function is implemented in a logic cell in a particular LAB, row, or column.

In FLEX 10K, FLEX 8000, FLEX 6000, and MAX 9000 devices, logic cells have "numbers" of the format `LC`*<number>*`_`*<LAB name>*, where *<number>* ranges from 1 to 8 and *<LAB name>* consists of the row letter and

column number of the LAB. In Classic, MAX 5000, and MAX 7000 devices, logic cells have numbers of the format LC<*number*>, where <*number*> may consist of both digits and letters.

FLEX 10K, FLEX 8000, and MAX 9000 devices have specialized logic cells, called I/O cells, on the periphery of the device.

logic cell Turbo Bit *see* Turbo Bit.

logic element *see* logic cell.

logic level The input and output logic levels of nodes and groups are defined with the following characters:

Character:	Logic Level:
0	Logic low (GND)
1	Logic high (VCC)
X	Undefined/Don't Care (not permitted for initialization)
Z	High impedance (no input to pin); e.g., used for the "output" part of a bidirectional pin when the "input" part of the pin is driving in.
0 to 9, A to F	Used for groups and interpreted as binary, decimal, hexadecimal, or octal values according to the current radix. The most significant bit is first; the least significant bit is last.

logic option An option that controls the logic synthesis process on one or more logic functions.

A variety of logic options are available. Logic option assignments can be applied to individual logic functions; a group of logic option assignments, called a logic synthesis style, can be applied to individual logic functions. A default logic synthesis style is also applied to the project as a whole. The logic cell Turbo Bit logic option can also be turned on or off on a device-by-device basis.

Logic options can also be assigned as parameters for a megafunction or macrofunction.

1. Some logic options are not available with standard synthesis; all logic options are available with multi-level synthesis.

2. A logic option is ignored if it does not apply to the current device family.

logic synthesis style A combination of logic synthesis option settings that are saved under a single name.

A logic synthesis style can be individually tailored for different device families, so that the logic synthesis option settings vary according to the architecture of the target device family.

If the global project logic synthesis style for your project is not fully defined, i.e., if the style specified with the **Global Project Logic Synthesis** command (Assign menu) uses a "default" setting for any logic option, the MAX+PLUS II Compiler will use the non-"default" setting for that logic option from the predefined, Altera-provided settings

for the Normal style. To view the settings for a predefined style, open the **Define Synthesis Style** dialog box, select the style in the *Style* box, and choose the **Use Default** button.

logical operator An operator that performs a logic operation on nodes, groups, or numbers.

AHDL logical operators are `NOT` (`!`), `AND` (`&`), `NAND` (`!&`), `OR` (`#`), `NOR` (`!#`), `XOR` (`$`), and `XNOR` (`!$`).

VHDL logical operators are `AND`, `NAND`, `OR`, `NOR`, `XOR`, and `NOT`.

Verilog HDL logical operators are `and` (`&&`) and `or` (`||`).

LPM *see* Library of Parameterized Modules.

LSB *see* least significant bit.

M

macrocell *see* logic cell.

macrofunction A high-level building block that can be used together with gate and flipflop primitives and/or megafunctions in MAX+PLUS II design files.

In general, Altera recommends using megafunctions in preference to equivalent macrofunctions in all new projects. Megafunctions are easier to scale to different sizes and offer more efficient logic synthesis and device implementation.

Altera provides a library of over 300 old-style macrofunctions in the **\maxplus2\max2lib** directory and its subdirectories created during installation. AHDL Include Files (**.inc**) for these macrofunctions are located in the **\maxplus2\max2inc** directory; VHDL Component Declarations for macrofunctions supported by VHDL are provided in the `maxplus2` package in the **altera** library, which is located in a subdirectory of the **\maxplus2\vhdl*nn*** directory, where ***nn*** is "87" or "93". On a UNIX workstation, the **maxplus2** directory is a subdirectory of the **/usr** directory.

To view the file that contains the logic for a macrofunction, select the macrofunction symbol in the Graphic Editor or macrofunction name in the Text Editor and choose **Hierarchy Down** (File menu).

MAX 5000 An Altera device family based on the first generation of Multiple Array MatriX architecture. This EPROM-based device family includes the EPM5032, EPM5064, EPM5128, EPM5128A, EPM5130, and EPM5192 devices.

MAX 7000, MAX 7000E, and MAX 7000S An Altera device family based on the second generation of Multiple Array MatriX architecture that includes MAX 7000, MAX 7000E, and MAX 7000S devices. These EEPROM-based devices include EPM7032, EPM7032V, EPM7064, EPM7064S, EPM7096, EPM7128E, EPM7128S, EPM7160E, EPM7192E, EPM7192S, EPM7256E, and EMP7256S devices.

MAX 7000S and 7000E devices are enhanced versions of MAX 7000 devices and are function-, pin-, and programming-file-compatible with MAX 7000 devices. MAX 7000E and MAX 7000S devices differ from MAX 7000 devices in that they offer up to six pin- or logic-driven Output Enable signals, fast input setup times to

logic cells, and multiple global Clocks with optional inversion. MAX 7000S devices also offer the additional capability of in-system programming via JTAG boundary-scan test circuitry.

☞ Altera strongly recommends using MAX 7000S and MAX 7000E devices rather than equivalent MAX 7000 devices for new designs.

MAX 9000 An Altera device family based on the third generation of Multiple Array MatriX architecture. These EEPROM-based devices include the EPM9560, EPM9560A, EPM9480, EPM9400, EPM9320, and EPM9320A devices.

MAX 9000A devices are enhanced versions of MAX 9000 devices, and are function-, pin-, and programming-file-compatible with MAX 9000 devices. MAX 9000A devices differ from MAX 9000 devices in that they offer an additional 16 bits for a user code.

MAX 9000 devices with the speed grade suffix "F" contain fixed programming algorithms, and therefore can be programmed with Serial Vector Format Files.

MAX+PLUS (DOS) Altera's DOS-based Multiple Array MatriX Programmable Logic User System. MAX+PLUS is a set of computer programs and hardware support products for designing and implementing custom logic circuits with Altera Classic and MAX 5000 devices. Graphic Design Files (**.gdf**) created for MAX+PLUS are automatically converted and processed with the MAX+PLUS II Compiler; AHDL Text Design Files (**.tdf**) are compiled directly. The MAX+PLUS II Programmer can program Classic and MAX 5000 devices with JEDEC Files (**.jed**) and Programmer Object Files (**.pof**) created by MAX+PLUS.

☞ MAX+PLUS is no longer offered by Altera. All new designs should be created with MAX+PLUS II.

MAX+PLUS II Message File (.mmf) A binary file (with the extension **.mmf**) created by MAX+PLUS II that contains messages issued by any MAX+PLUS II application or command that runs as a background process, e.g., the Compiler and Programmer. This file is used to display messages in the Message Processor and to locate messages in design and ancillary files.

maxplus2.idx file A text file, created automatically when you save a file, that maps filenames with more than eight characters to 8-character filenames.

MAX+PLUS II creates a **maxplus2.idx** file in each directory where you save a file that contains filenames with more than eight characters. The file is automatically updated each time you save a file with a long filename.

maxplus2.ini file A text file, created during installation, that contains the parameters that affect the way MAX+PLUS II applications operate. This file continuously records the options that you set in one session, so that they are automatically set for the next session.

MegaCore and OpenCore megafunctions MegaCore and OpenCore megafunctions are pre-verified HDL design files for complex system-level functions that can be purchased from Altera. These pre-tested megafunctions are optimized for

FLEX 10K, FLEX 8000, FLEX 6000, MAX 9000, and MAX 7000 device architectures. Altera MegaCore megafunctions consist of several different design files. A post-synthesis AHDL design file is used for design implementation (i.e., fitting) in the target Altera device. In addition, VHDL or Verilog HDL functional simulation models are supplied for design and debugging with standard EDA simulation tools.

OpenCore megafunctions are MegaCore functions that you can use and evaluate before purchasing full support. If you purchase full support, you can generate programming files and EDIF, VHDL, and Verilog HDL output files for post-compilation simulation with other EDA tools.

Altera provides a library of megafunctions, including OpenCore megafunctions, in the **\maxplus2\max2lib\mega_lpm** directory. (On a UNIX workstation, the **maxplus2** directory is a subdirectory of the **/usr** directory). VHDL Component Declarations for megafunctions supported by VHDL are provided in the `megacore` package in the **altera** library, which is located in the **\maxplus2\vhdl*nn*** directory, where ***nn*** is "87" or "93".

If your authorization code for a MegaCore megafunction includes permission to view the source design file, you can view the file by selecting the megafunction symbol in the Graphic Editor or megafunction name in the Text Editor and choosing **Hierarchy Down** (File menu).

megafunction A complex or high-level building block that can be used together with gate and flipflop primitives and/or old-style macrofunctions in MAX+PLUS II design files.

Altera provides a library of megafunctions, including functions from the Library of Parameterized Modules (LPM) version 2.1.0, in the **\maxplus2\max2lib\mega_lpm** directory created during installation. AHDL Include Files (**.inc**) for these megafunctions are also located in the **\maxplus2\max2lib\mega_lpm** directory. VHDL Component Declarations for LPM functions and other megafunctions are provided in the `lpm_components` package in the **lpm** library, and the `megacore` package in the **altera** library, respectively. Both of these libraries are located in subdirectories of the **\maxplus2\vhdl*nn*** directory, where ***nn*** is "87" or "93". (On a UNIX workstation, the **maxplus2** directory is a subdirectory of the **/usr** directory.)

To view the file that contains the logic for a megafunction, select the megafunction symbol in the Graphic Editor or megafunction name in the Text Editor and choose **Hierarchy Down** (File menu).

memory bit and **memory word** A memory bit is an individual memory address in a memory (i.e., RAM or ROM) block.

A memory word is a group of memory bits in a RAM or ROM block.

For example, the `content5_[4..0]` memory word defines a byte of memory in which the individual memory bits are `content5_4`, `content5_3`, `content5_2`, `content5_1`, and `content5_0`.

Memory Initialization File (.mif) An ASCII file (with the extension **.mif**) that specifies

the initial content of a memory block (RAM or ROM), i.e., the initial values for each address. This file is used during project compilation and/or simulation.

Memory Initialization Output File (.mio) An ASCII file (with the extension **.mio**) that is generated when the Compiler creates a Text Design Export File (**.tdo**) for a project. A TDO File that implements RAM or ROM always has an MIO File for each memory segment.

An MIO File specifies the memory addresses and values used to initialize a RAM or ROM segment, similar to the information in a Memory Initialization File (**.mif**).

You can rename an MIO File as an MIF and use it with a TDO File that has been saved as a Text Design File (**.tdf**).

memory segment or **segment** The physical implementation of memory (i.e., RAM or ROM) in a device. A memory segment contains a sequence of memory bits corresponding to an address range.

In FLEX 10K devices, a memory segment consists of that portion of a bit-slice of a memory which is implemented in a single embedded cell. Each embedded cell implements up to 256 bits of memory. Multiple memory segments may be needed to create a single memory block.

Message Text File (.mtf) An ASCII file (with the extension **.mtf**) that contains the text of messages shown in a Message Processor window.

MIF *see* Memory Initialization File.

MMF *see* MAX+PLUS II Message File.

most significant bit (MSB) The bit of a binary number that contributes the greatest quantity to the value of that number, and the first member in a bus or group name. For example, the MSB for a bus named `a[31..0]` is `a[31]`.

MSB *see* most significant bit.

MTF *see* Message Text File.

multi-device FLEX chain A series of devices through which configuration data is passed from device to device using the sequential Passive Serial configuration scheme.

The MAX+PLUS II Programmer can configure multiple FLEX 6000, FLEX 8000, or FLEX 10K devices in a multi-device FLEX chain.

multi-device JTAG chain A series of devices through which programming and/or configuration data are passed from device to device via the Joint Test Action Group (JTAG) Boundary-Scan Test (BST) circuitry.

The MAX+PLUS II Programmer can program or configure multiple MAX 7000S, MAX 9000, and FLEX 10K devices in a multi-device JTAG chain. The JTAG chain can contain any combination of Altera and non-Altera devices that comply with the IEEE 1149.1 JTAG specification, including some FLEX 8000 devices.

MAX+PLUS II can also generate Jam Files (**.jam**) and Serial Vector Format Files (**.svf**) that support programming for one or more MAX 7000S and MAX 9000 devices in a JTAG chain. SVF files can be used in Automated Test Equipment (ATE)-type programming environments; Jam Files in

embedded processor-type programming environments.

multi-level synthesis Logic synthesis that takes advantage of all available logic options, including all options listed in the **Define Synthesis Style** and **Advanced Options** dialog boxes (Assign menu). This type of logic synthesis can handle projects with extremely complex logic, without requiring user intervention to achieve a fit.

Multi-level synthesis can be selected with the **Global Project Logic Synthesis** dialog box (Assign menu). This type of synthesis is available only for the MAX 5000, MAX 7000, MAX 9000, FLEX 6000, FLEX 8000, and FLEX 10K device families; it is the only type of synthesis available for FLEX 6000, FLEX 8000, and FLEX 10K projects.

N

name characters The characters A to Z, a to z, 0 to 9, slash (/), dash (-), and underscore (_) are legal for MAX+PLUS II breakpoint, chip, clique, file, group (bus), node, parameter, pin, pinstub, probe, logic synthesis style, and quoted and unquoted symbolic names, with the exceptions listed below. Case is significant only in Verilog HDL files.

Item:	Name Character Exception:
filename	No slash (/) is permitted. Case is significant on UNIX workstations.
single-range group (bus) name	No slash (/) is permitted; the bus identifier cannot end with a digit. The name is followed by a range of numbers or arithmetic expressions in brackets. The start and end of the range are separated by two periods. For example, group `a[3..1]` consists of the nodes `a3`, `a2`, and `a1`. In Graphic Editor files only, sequential bus names can also include a series of single-range bus names. For example, `a[8..0],d[6..4]`.
dual-range group (bus) name	Same as single-range group names, with two ranges of numbers or arithmetic expressions in brackets. For example, `a[6..3][4..0]`.
sequential group (bus) name	The name consists of a series of comma-separated node names enclosed in parentheses. For example, group `(a, b, c)` consists of the nodes `a`, `b`, and `c`. In Graphic Editor files, parentheses are not used.
unquoted symbolic name (AHDL)	No dash (-) is permitted. Names cannot consist entirely of digits. AHDL keywords cannot be used.
Verilog HDL identifiers	No slash (/) or dash (-) is permitted. Names cannot begin with a digit. Case is significant. Verilog HDL keywords cannot be used.

Item:	Name Character Exception:
VHDL names	No slash (/) or dash (-) is permitted. The name must start with a letter, cannot end with an underscore (_), and cannot contain two underscores (_ _) in a row. VHDL keywords cannot be used.
ACF names	Names that contain slash (/), dash (-), vertical bar (|), colon (:), and/or period (.) characters must be enclosed in double quotation marks (").

net ID number *see* symbol ID number.

network A group of interconnected node and/or bus lines, including nodes or buses that are connected by name only.

node A node represents a wire carrying a signal that travels between different logical components of a design file. In Verilog HDL, nodes are called "nets."

In the Graphic Editor files, nodes are represented as lines; in text files, they are symbolic names; in Waveform Editor files, they are waveforms.

Node Database File (.ndb) A file that contains the database of project node names, which supports resource and probe assignment edits with Assign menu commands and the Floorplan Editor. The Compiler Netlist Extractor and Database Builder modules of the Compiler generate a Node Database File for a project during project processing.

1. If you turn on the Compiler's **Preserve All Node Name Synonyms** command (Processing menu) before compilation, this file will not contain all possible forms of the project node names.

2. If you accidentally delete this file, you must recompile the project before you can use most Assign menu commands and Floorplan Editor functions. In addition, only pins are visible in the Floorplan Editor if a Node Database File is created with the **Project Save & Check** command (File menu): a full compilation is required to make buried nodes visible in the Floorplan Editor.

node or **net name** The name given to a signal in a design file. A node or net name can contain up to 32 of the following name characters: A to Z, a to z, 0 to 9, slash (/), dash (-), and underscore (_). Hierarchical node names can contain 128 characters, including vertical bar (|), colon (:), and period (.). Case is not significant.

Some restrictions apply to names in VHDL Design Files (**.vhd**), Verilog Design Files (**.v**), and unquoted port and symbolic names in AHDL Text Design Files (**.tdf**).

node type The type of logic that drives a node or group in a Waveform Design File (**.wdf**) or Vector File (**.vec**). Four logic types are defined:

Type:	Meaning:
INPUT	Node or group is driven by an input pin.

Type:	Meaning:
COMB	Node or group is fed by combinatorial logic, e.g., an AND gate.
REG	Node or group is fed by a register (implemented with a logic cell on the device).
MACH	Node is fed by a state machine.

Normal logic synthesis style The Altera-provided style that directs the Logic Synthesizer to optimize your project for minimum silicon resource usage.

The Normal style attempts to use device resources as efficiently as possible, without adding excessive timing delays.

To display the settings for this style, select the style in the **Define Synthesis Style** dialog box, which is available through the **Logic Options** or **Global Project Logic Synthesis** dialog boxes (Assign menu).

If the global project logic synthesis style for your project is not fully defined, i.e., if the style specified with the **Global Project Logic Synthesis** command (Assign menu) uses a "default" setting for any logic option, the MAX+PLUS II Compiler will use the non-"default" setting for that logic option from the predefined, Altera-provided settings for the Normal style. To view the settings for a predefined style, open the **Define Synthesis Style** dialog box, select the style in the *Style* box, and choose the **Use Default** button.

O

object-by-object selection The process of selecting multiple non-contiguous objects. The first object is selected by clicking Button 1 on it. You can add or remove objects to the selection by pressing Shift while clicking on them with Button 1.

In the Graphic Editor, object-by-object selection can be used to select graphics and/or text blocks; in the Waveform Editor, to select nodes and groups; in the Floorplan Editor, to select pins, nodes, logic cells, or assignment bins; and in the Hierarchy Display, to select file icons.

In the Graphic Editor, multiple objects in a rectangular area can be selected and added to an existing selection by pressing Shift while dragging Button 1.

octal The base 8 number system (radix). Octal digits are 0 though 7.

Octal numbers are indicated with the following notation:

Language	Notation
AHDL	**O"***<series of digits 0 to 7>***"** *or* **Q"***<series of digits 0 to 7>***"**
VHDL	**8#***<series of digits 0 to 7>***#**
Verilog HDL	**'o***<series of digits 0 to 7>*

Examples:

```
Q"4671223" (AHDL)
8#4671223# (VHDL)
'o4671223 (Verilog HDL)
```

one-hot encoding A type of binary coding in which one and only one bit of a value is set to 1. For example, the four legal values 0001, 0010, 0100, and 1000 together

comprise a "one-hot" code sample because in each of these four values a single bit is set to 1.

You can manually implement one-hot encoding. In addition, the **Global Project Logic Synthesis** command (Assign menu) includes a *One-Hot State Machine Encoding* option to allow the Compiler to automatically implement one-hot encoding for the entire project. Altera strongly recommends using the *One-Hot State Machine Encoding* option rather than manual one-hot encoding.

This option is available in both multi-level and standard synthesis. It is ignored if it does not apply to the current device family.

OrCAD Library File (.lib) A binary file (with the extension **.lib**) containing information that describes how symbols are displayed in OrCAD Schematic Files (**.sch**).

The MAX+PLUS II Graphic Editor uses an OrCAD-generated OrCAD Library File to import an OrCAD Schematic File. The OrCAD Library File for each OrCAD Schematic File should contain all libraries for the OrCAD symbols used in the schematic. This OrCAD Library File must also be copied to same directory as the OrCAD Schematic File.

OrCAD Schematic File (.sch) A schematic design file (with the extension **.sch**) created with the OrCAD Draft schematic editor. You can open and edit an OrCAD Schematic File in MAX+PLUS II and save it as both a Graphic Design File (**.gdf**) and an OrCAD Schematic File (**.sch**). An OrCAD Schematic File can also be compiled directly by the MAX+PLUS II Compiler.

oscillation An unstable logic level on a signal. When the Simulator is in timing or linked simulation mode, you can specify the time period that constitutes an oscillation and monitor the project for signals that do not stabilize within the defined period. When the Simulator is in functional simulation mode, you can monitor the project for nil-period oscillation only.

Output Enable A high logic level on the Output Enable signal enables the output.

In MAX 7000 devices (not including MAX 7000E devices), the signal from the active-low global Output Enable pin must be inverted and connected to the active-high Output Enable input of the `TRI` primitive. In all other device families, either active-high or active-low polarity can be used.

In MAX 9000 devices, the Fitter automatically inserts additional `LCELL` primitives to provide the correct polarity for a non-global Output Enable pin or an Output Enable signal driven by a logic cell.

P

parameter or parameterized A parameter is an attribute of a megafunction or macrofunction that determines the logic created or used to implement the function, i.e., a characteristic that determines the size, behavior, or silicon implementation of a function. The parameter information can be used to determine the actual primitives and other subdesigns needed to implement the logic of the function.

A parameterized function is a function whose behavior is controlled by one or more parameters. Some logic functions,

such as the functions in the Library of Parameterized Modules (LPM), are inherently parameterized and require parameter values to be assigned.

Parameters can be assigned to any individual instance of a megafunction in MAX+PLUS II to control its size or implementation. Some parameters can also be applied to old-style macrofunctions to determine their style of implementation. MAX+PLUS II also allows you to assign global, project-wide default values for parameters.

parameterized module A logic function that uses parameters to achieve scalability, adaptability, and efficient silicon implementation. MAX+PLUS II supports a variety of parameterized modules (also called "parameterized functions"), including functions belonging to the Library of Parameterized Modules (LPM).

LPM functions provide architecture-independent design entry for all MAX+PLUS II-supported devices. The MAX+PLUS II Compiler includes built-in compilation support for LPM functions used in schematic, AHDL, VHDL, Verilog HDL, and EDIF input files.

pin A pin is an actual input or I/O pin on an Altera device.

In Graphic Editor files, a pin is represented by an `INPUT`, `INPUTC`, `OUTPUT`, `OUTPUTC`, `BIDIR`, or `BIDIRC` symbol. In a Text Design File (**.tdf**), a pin is represented as an `INPUT`, `OUTPUT`, or `BIDIR` port. In a VHDL Design File (**.vhd**), a pin is represented as an `IN`, `OUT`, or `INOUT` port. In a Verilog Design File (**.v**), a pin is represented as an `input`, `output`, or `inout` port. In a Waveform Design File (**.wdf**), a pin is represented as a node with an input, output, or bidirectional I/O type and a pin input, registered, or combinatorial node type.

You can assign a logic function to a specific pin number. You can also assign a logic function to a row or a column to ensure that the function is implemented in a pin on a particular row or column.

pin number A number used to assign an input or output signal in a design file, which corresponds to the pin number on an actual device.

Both letters and digits are used to specify pin numbers for PGA-package devices.

pinstub In the Graphic and Symbol Editors, a pinstub is the location on the boundary of a symbol represented by an "**x**" in a Symbol File (**.sym**) and a name that represents an input or output of the primitive or of the megafunction or macrofunction design file that the symbol represents. A line (node) drawn in a schematic must connect to this pinstub to be recognized by the Compiler as a connection between the logic in the current file and the logic in the primitive, megafunction, or macrofunction.

You can specify whether or not to use an optional pinstub when you edit a symbol instance in a Graphic Editor file.

Pinstubs in Graphic Editor files are synonymous with ports in AHDL Function Prototypes and VHDL Component Declarations. They are also synonymous with ports listed in the Subdesign Sections of lower-level Text Design Files (**.tdf**); in Entity Declarations of lower-level VHDL Design Files (**.vhd**); and in Module

Declarations, Module Instantiations, and Gate Instantiations of Verilog Design Files (**.v**).

pinstub name A symbolic name that identifies an input or output of a logic function.

In the Symbol Editor, the "visible" pinstub name appears both inside and outside of the symbol. This "visible" pinstub name can be an abbreviation or an alias for the "full" pinstub name, which represents the full name of the original input, output, or bidirectional pin in a mega- or macrofunction design file or primitive Function Prototype.

You can specify whether or not to display the "visible" pinstub name in a Graphic Editor file when you create a pinstub in the Symbol Editor. The use or non-use of a particular pinstub (and hence its visibility) can be customized when you edit a symbol instance in the Graphic Editor with **Edit Ports/Parameters** (Symbol menu).

Pinstubs in Graphic Editor files are synonymous with ports in AHDL Function Prototypes and VHDL Component Declarations. They are also listed in the Subdesign Sections of lower-level Text Design Files (**.tdf**); in Entity Declarations of lower-level VHDL Design Files (**.vhd**); and in Module Declarations, Module Instantiations, and Gate Instantiations of Verilog Design Files (**.v**).

PLF *see* Programmer Log File.

PLS-ES A PC-based MAX+PLUS II development system, which is automatically provided on a site license when you purchase any PC-based MAX+PLUS II development system.

PLS-ES development systems include the following MAX+PLUS II applications and features:

- Hierarchy Display
- Graphic, Symbol, and Text Editors
- Compilation support for Classic, MAX 5000, MAX 7000/7000E/7000S, EPF8282, EPF8452, EPM9320, and EPF10K10 devices
- EDIF Interfaces (input and output)
- Verilog HDL and VHDL output
- Timing Analyzer
- Message Processor
- Programmer

POF *see* Programmer Object File.

port A symbolic name that represents an input or output of a primitive or of a design file.

In AHDL, a port name in the Subdesign Section represents an input or output of the current file. This port name also appears in the Function Prototype for the function. When an instance of a primitive or lower-level design file is implemented with an Instance Declaration or an in-line reference, its ports are used to connect it to other functions in the TDF. After an instance is declared, its inputs and outputs are expressed as names in the format *<instance name>*.*<port name>* in the Logic Section. When an in-line reference is used, either named port association or positional port association can be used to connect the function's ports to other functions in the TDF.

In VHDL, a port name in the Entity Declaration represents an input or output of the current file. When an instance of a primitive or lower-level design file is implemented with a Component

Instantiation, its ports are connected to signals with Port Map Aspects.

In Verilog HDL, a port in a Module Declaration represents an input or output of the current file. When an instance of a lower-level design file is implemented with a Module Instantiation, its ports are connected by order or by name to the Module Declaration ports of the module being instantiated. Similarly, when a primitive is implemented with a Module Instantiation, its ports are used to connect it by order of other functions in the file. Verilog HDL gate primitives also contain ports (called 'terminals"); when a gate primitive is implemented with a Gate Instantiation, its terminals are connected by order to the terminals of the gate being instantiated.

A port name in an AHDL Subdesign Section, VHDL Entity Declaration, or Verilog HDL Module or Gate Declaration is synonymous with a pin name in a Graphic Design File (**.gdf**) or Waveform Design File (**.wdf**). A port name that is appended to an instance name is synonymous with the full pinstub name in an instance of a symbol in a Graphic Editor file.

Preset An input signal that asynchronously sets the output of a register to a logic high (`1`), regardless of other inputs.

primitive One of the basic functional blocks used to design circuits with MAX+PLUS II software. Primitives are used in Graphic Design Files (**.gdf**), AHDL Text Design Files (**.tdf**), VHDL Design Files (**.vhd**), and Verilog Design Files (**.v**).

Graphic Editor primitives include buffers, flipflops, a latch, input and output primitives, and logic primitives. Primitive symbols for Graphic Editor files are provided in the **\maxplus2\max2lib\prim** directory created during installation.

AHDL, VHDL, and Verilog HDL primitives, which include buffers, flipflops, and a latch, are a subset of the primitive symbols used in Graphic Editor files. Other functions are represented by logical operators, ports, and other constructs. Function Prototypes for AHDL primitives are built into the MAX+PLUS II software; Component Declarations for VHDL primitives are provided in the `maxplus2` package in the **\maxplus2\max2vhdl*nn*\altera** directory, where ***nn*** is "87" or "93".

On a UNIX workstation, the **maxplus2** directory is a subdirectory of the **/usr** directory.

primitive array A single primitive that is connected to two or more buses in order to represent multiple primitives.

probe A unique name assigned to any node, e.g., the input or output of a primitive, megafunction, or macrofunction which can be used instead of the full hierarchical node name throughout MAX+PLUS II. A probe name thus provides a short name to identify a node.

product term Two or more factors in a Boolean expression combined with an `AND` operator constitute a product term, where "product" means "logic product."

Programmer Log File (.plf) An ASCII file (with the extension **.plf**) generated by the Programmer that records programming session commands and messages.

Programmer Object File (.pof) A binary file (with the extension **.pof**) generated by the Compiler's Assembler module. This file contains the data used by the MAX+PLUS II Programmer to program an Altera device. The MAX+PLUS II Programmer can optionally save functional test vectors in a POF.

programming file A file containing data for programming Altera devices. Both the MAX+PLUS II Compiler and Programmer can generate programming files. The following programming file formats are available in MAX+PLUS II:

- FLEX Chain File (**.fcf**)
- Hexadecimal (Intel-Format) File (**.hex**)
- Jam File (**.jam**)
- JEDEC File (**.jed**)
- JTAG Chain File (**.jcf**)
- Programmer Object File (**.pof**)
- Raw Binary File (**.rbf**)
- Serial Bitstream File (**.sbf**)
- Serial Vector Format File (**.svf**)
- SRAM Object File (**.sof**)
- Tabular Text File (**.ttf**)

POFs, SOFs, JEDEC Files, JCFs, and FCFs are used to program or configure devices with the MAX+PLUS II Programmer; test vectors for functional testing can be saved in POFs and JEDEC Files. All other file formats are used to program or configure devices in other environments.

JEDEC Files generated by A+PLUS and PLDshell Plus software can also be used to program Classic devices. The Programmer can save data read from an examined device in POF or JEDEC File format.

project A project consists of all files that are associated with a particular design, including all subdesign files and related ancillary files created by the user or by MAX+PLUS II software. The project name is the same as the name of the top-level design file in the project, without the filename extension.

MAX+PLUS II performs compilation, simulation, timing analysis, and programming on only one project at a time.

propagation delay The time required for any signal transition to travel between pins and/or nodes in a device.

R

radix A number base. Group logic level and numerical values are entered and displayed in binary, decimal, hexadecimal, or octal radix in MAX+PLUS II.

RAM Random-access memory. You can implement RAM with Embedded Array Blocks (EABs) in the FLEX 10K device family, and with arrays of flipflops or latches in other device families.

range A sequence of numbers or arithmetic expressions that define the width of a group (bus). A range is enclosed in brackets; the most significant bit (MSB) of the range is shown first; the least significant bit (LSB) is shown last. The start and end of the range are separated by two periods in the Graphic Editor and in AHDL, and by a colon in Verilog HDL.

Example: group `a[2..0]` consists of the nodes `a2`, `a1`, and `a0`; the MSB is `a2`; and the LSB is `a0`.

Raw Binary File (.rbf) A binary file (with the extension **.rbf**) containing configuration data for FLEX 6000, FLEX 8000, and FLEX 10K devices. This file is the binary equivalent of a Tabular Text File (**.ttf**).

You can create RBFs that support Passive Parallel Synchronous (PPS), Passive Parallel Asynchronous (PPA), and Passive Serial (PS) configuration schemes in MAX+PLUS II.

register *see* flipflop.

register packing A feature of logic cells in MAX 9000 and FLEX 10K devices that allows two logic functions—a combinatorial logic function and a register with a single data input—to be implemented in the same logic cell.

You can manually implement register packing by assigning two logic functions to the same logic cell. In addition, the **Global Project Logic Synthesis** command (Assign menu) includes an *Automatic Register Packing* option to allow the Compiler to automatically implement register packing for appropriate pairs of logic functions. Altera strongly recommends using the *Automatic Register Packing* option rather than manual logic cell assignments to implement register packing.

This option is available in both multi-level and standard synthesis. It is ignored if it does not apply to the current device family.

registered feedback Feedback that is the output of a flipflop or latch.

registered output The output of a flipflop or latch, which can feed an output pin on the device.

registered performance The minimum required Clock period and the maximum Clock frequency for a circuit, which can be calculated in the MAX+PLUS II Timing Analyzer.

The Clock period equals the maximum delay from the Q output of a flipflop to the D or Clock Enable input of a flipflop, plus the internal setup time and propagation delay through the flipflop. Clock skew calculations may increase the Clock period. The Clock frequency equals (1/Clock period).

The Timing Analyzer does not calculate registered performance for a signal path that passes through the primary (data) input to a flipflop.

Report File (.rpt) An ASCII text file (with the extension **.rpt**), generated by the Compiler's Fitter module, that shows how device resources are used by the project. If a module preceding the Partitioner generates an error, this file is not generated. If the Partitioner generates an error, the Report File is generated in most cases.

Reset An active-high input signal that asynchronously resets the output of a register to a logic low (`0`) or a state machine to its initial state, regardless of other inputs.

resource A resource is a portion of an Altera device that performs a specific, user-defined task (e.g., pins, logic cells).

resource assignment An assignment of a logic function in a project to a particular pin, logic cell, I/O cell, embedded cell, logic array block (LAB), embedded array block (EAB), row, column, or chip. This type of resource assignment assigns a logic function to a physical resource in a device.

A resource assignment can also consist of a clique, logic option, connected pin, timing requirement, or local routing assignment to a particular logic function in a project. This type of resource assignment assigns a compilation resource to a logic function.

row A horizontal line of LABs connected by a row FastTrack Interconnect path in a FLEX 6000, FLEX 8000, FLEX 10K, or MAX 9000 device.

RS-232 port *see* COM port.

S

SCF *see* Simulator Channel File.

SDF Output File *see* Standard Delay Format Output File.

secondary input The Clock, Preset, synchronous and asynchronous Reset (Clear), and synchronous and asynchronous Load inputs to a register or a state machine in a design file.

Security Bit A bit that prevents an EPROM- or EEPROM-based Altera device from being interrogated. This bit also prevents EPROM-based Altera devices from being inadvertently reprogrammed.

The Security Bit can be turned on or off for each device in a project, or for the entire project.

segment *see* memory segment.

Serial Bitstream File (.sbf) An ASCII file (with the extension **.sbf**) that contains the data for configuring a FLEX 6000, FLEX 8000, or FLEX 10K device with the BitBlaster from a system prompt. This file can be generated with the **Combine Programming Files** command (File menu) in the Compiler or the Simulator.

Serial Vector Format File (.svf) An ASCII file (with the extension **.svf**) that stores programming data for programming one or more fixed algorithm devices in Automated Test Equipment (ATE)-type programming environments. Altera's MAX 7000S and MAX 9000 devices can be programmed with SVF Files. The JTAG chain can contain any other device that complies with the IEEE 1149.1 JTAG specification, including FLEX 10K, FLEX 6000, and some FLEX 8000 devices. You can create SVF Files with the **Create Jam or SVF File** command (File menu) in the Programmer or the Compiler.

setup time On a flipflop, the setup time is the minimum time interval between the application of a signal at the input pin that feeds the data or Clock Enable input and a low-to-high transition at the input pin that feeds the Clock input of the flipflop or the Latch Enable input of the latch.

On a latch, the setup time is the minimum time interval between the application of a signal at the input pin that feeds the data input of a latch and a low-to-high transition at the input pin that feeds the Latch Enable input of the latch. (Setup and hold time analysis for latches is available only for MAX 5000 devices. In other device families, latches are implemented using combinatorial logic with feedback.)

On an asynchronous RAM block, the setup time is the minimum time interval between the application of a signal at the input pin that feeds the data or address inputs and a low-to-high or high-to-low transition at the input pin that feeds the Write Enable input of the RAM block.

Internal setup times for flipflops, latches, and asynchronous RAM, which are not user-defined, similarly constrain signals that are generated within the device.

shared local interconnect Dedicated connection paths on FLEX 6000 devices that allow signals to travel quickly between logic cells in the same LAB or adjacent LABs, or between logic cells on the periphery of the device and I/O pins. Shared local interconnect is the fastest interconnect available in FLEX 6000 devices.

SIF *see* Simulator Initialization File.

Simulator Channel File (.scf) A graphical waveform file (with the extension **.scf**) that is both an input and an output to the Simulator. This file contains a waveform representation of the vector values on the input nodes that drive simulation, as well as the buried and output nodes to be simulated. Waveforms in the file represent high (1), low (0), high-impedance (Z), and undefined (X) logic levels.

An SCF can be created and viewed in the Waveform Editor; the Simulator also automatically creates and updates an SCF during simulation. An SCF can also be used to provide the vector inputs for functional testing in the Programmer.

Simulator Initialization File (.sif) A file (with the extension **.sif**) that saves all node, group, and memory values, including initialized values entered with the Simulator's **Initialize Nodes/Groups** and **Initialize Memory** commands (Initialize menu) or the Command File (**.cmd**) `GROUP INIT` and `NODE INIT` commands. The SIF allows you to reuse a previously saved set of node and group values.

Simulator Netlist File (.snf) A binary file (with the extension **.snf**) that contains the data for functional simulation, timing simulation or timing analysis, or linked multi-device simulation. Three optional Compiler modules create the different types of SNFs that contain the information required for different simulation modes and/or timing analysis:

- The Timing SNF Extractor generates a timing SNF that contains all data required for timing simulation and full timing analysis.
- The Functional SNF Extractor generates a functional SNF that contains all data required for functional simulation.
- The Linked SNF Extractor generates a linked SNF that combines timing and/or functional data from the timing, functional, and/or linked SNFs for other previously compiled projects. If all of the combined SNFs are timing SNFs, a linked SNF can also be used for full timing analysis.

Only one type of SNF can exist at any particular time for the same project.

single-range group (or bus) name The name of a group (or bus) of up to 256 nodes, consisting of an identifier with up to 32 name characters, followed by a range of numbers or arithmetic expressions in brackets. The start and end of the range are

separated by two periods. Each number in the sequence represents an individual node (or "bus bit"). The identifier cannot end with a digit.

Example: group `a[4..1]` consists of the nodes `a4`, `a3`, `a2`, and `a1`.

In a Graphic Editor files, a sequential bus name can also include one or more single-range bus names in a series. The first node of the series or the first node in the first range is the most significant bit of the bus; the last node of the series or the last node in the last range is the least significant bit.

Example: `a[8..0], b1, dout[6..4]`

SNF *see* Simulator Netlist File.

SOF *see* SRAM Object File.

source node A node that is tagged (designated) as the source of a signal for the purpose of timing analysis. A source node is tagged with the **Timing Analysis Source** command (Utilities menu), and can be any node that is the output of a primitive, megafunction, macrofunction, or I/O pin.

spike *see* glitch.

SRAM Object File (.sof) A binary file (with the extension **.sof**), generated by the Compiler's Assembler module, that contains the data for configuring an Altera FLEX 6000, FLEX 8000, or FLEX 10K device.

Standard Delay Format Output File (.sdo) An optional output file (with the extension **.sdo**) containing timing delay information that allows you to perform back-annotation for simulation with VHDL simulators that use simulation libraries that are compliant with VITAL version 2.2b and version 3.0 (VITAL 95); back-annotation for simulation in Verilog HDL simulators; and timing analysis and resynthesis with EDIF simulation and synthesis tools.The Standard Delay Format (SDF) is an industry-standard format.

The MAX+PLUS II Compiler's EDIF, VHDL, and Verilog Netlist Writer modules of the MAX+PLUS II Compiler can generate SDF Output Files in SDF version 2.1 or 1.0 format.

standard synthesis Logic synthesis that includes the following logic options:

- Fast I/O
- Global Signal
- Hierarchical Synthesis
- Insert Additional Logic Cell
- Minimization (Full and Partial)
- NOT Gate Push-Back
- Parallel Expanders
- Slow Slew Rate
- SOFT Buffer Insertion
- Turbo Bit (including logic cell Turbo Bit)
- Use LPM for AHDL Operators
- XOR Synthesis

Standard synthesis includes only these logic options; other options listed in the **Define Synthesis Style** and **Advanced Options** dialog boxes (Assign menu) are available only in multi-level synthesis. This type of logic synthesis is available only for the Classic, MAX 5000, MAX 7000, and MAX 9000 device families.

state bit An output of a flipflop used by a state machine to store one bit of the value of the state machine.

state machine A sequential circuit that advances through a number of states. A state machine can be defined in a Waveform Design File (**.wdf**), State Machine File (**.smf**), Vector File (**.vec**), VHDL Design File (**.vhd**), Verilog Design File (**.v**) or in a State Machine Declaration in an AHDL Text Design File (**.tdf**).

sub-project *see* super-project.

subdesign A lower-level design file in a MAX+PLUS II project, i.e., an Altera-provided or user-created megafunction or macrofunction.

Altera provides libraries of mega- and macrofunctions in the **mega_lpm** and **mf** subdirectories of the **\maxplus2\max2lib** directory. AHDL Include Files (**.inc**) for these functions are located in the **\maxplus2\max2lib\mega_lpm** and **\maxplus2\max2inc** directories, respectively. Component Declarations for functions supported by VHDL are provided in the `maxplus2` and `megacore` packages in the **altera** library and the `lpm_components` package in the **lpm** library. Both of these libraries are located in the **\maxplus2\vhdl*nn*** directory, where ***nn*** is "87" or "93." Megacore megafunctions must be purchased from Altera before you can use them in your design(s). (On a UNIX workstation, the **maxplus2** directory is a subdirectory of the **/usr** directory.)

subdesign name or **entity name** A name that represents the name of a subdesign. In AHDL, the subdesign name is a quoted or unquoted symbolic name that must be the same as the Text Design File (**.tdf**) filename. In VHDL, the entity name is an identifier.

Unquoted subdesign name (AHDL):

Maximum length:	32 characters
Legal characters:	`a-z`, `A-Z`, `0-9`, and underscore (_)
	An unquoted subdesign name cannot be a reserved AHDL identifier or keyword.

Quoted subdesign name (AHDL):

Maximum length:	32 characters
Legal characters:	`a-z`, `A-Z`, `0-9`, dash (`-`), and underscore (_)

Identifier (VHDL):

Maximum length:	32 characters
Legal characters:	a-z, `A-Z`, 0-9, and underscore (_)
	An identifier cannot begin with a digit or an underscore, cannot end with an underscore, and cannot have two underscores (_ _) in succession. It cannot be a keyword.

Identifier (Verilog HDL):

Maximum length:	32 characters
Legal characters:	`a-z`, `A-Z`, `0-9`, and underscore (_)
	An identifier cannot begin with a digit. Identifiers are case-sensitive. Verilog HDL keywords cannot be used.

☞ In the UNIX workstation environment, filenames and hence subdesign names are case-sensitive.

sum-of-products A Boolean expression is said to be in sum-of-products form if it consists of product terms combined with the OR operator.

super-project and **sub-project** A super-project consists of a top-level design file that contains symbols or instances representing multiple, individual projects.

A sub-project is any individual project that serves as part of a super-project. You can use any super-project as a sub-project in a higher-level super-project.

SVF File *see* Serial Vector Format File.

Symbol File (.sym) A graphic file (with the extension **.sym**) created by the Symbol Editor or the Compiler Netlist Extractor module of the Compiler. This file represents a design file (i.e., megafunction or macrofunction) or MAX+PLUS II primitive with the same name and can be used in Graphic Design Files (**.gdf**).

When the Compiler's Linked SNF Extractor module is turned on, a symbol in a Graphic Editor file represents a Simulator Netlist File (**.snf**), rather than a design file, during compilation.

symbol ID number or **net ID number** A number that uniquely identifies every node and symbol in a design file.

In the Graphic Editor, this number appears inside the bottom left corner of a symbol and reflects the order in which symbols are entered in a Graphic Editor file. In other types of design files, the Compiler assigns ID numbers to nodes when the project is compiled. In the Hierarchy Display window, the name of each lower-level design file is appended with a colon (:) plus the ID number or an AHDL,VHDL, or Verilog HDL mega- or macrofunction instance name.

T

Table File (.tbl) An ASCII file (with the extension **.tbl**) that contains a tabular-format list of all input vectors and output logic levels in the current Vector File (**.vec**) or Simulator Channel File (**.scf**). The Table File can be generated in the Simulator or Waveform Editor.

Tabular Text File (.ttf) An ASCII text file in tabular format (with the extension **.ttf**) containing configuration data for FLEX 6000, FLEX 8000, and FLEX 10K devices. A TTF contains the decimal equivalent of a Raw Binary File (**.rbf**).

The MAX+PLUS II Compiler automatically creates TTFs containing configuration data for the sequential Passive Parallel Synchronous (PPS), Passive Parallel Asynchronous (PPA), and Passive Serial (PS) configuration schemes for FLEX 8000 devices, the PS configuration scheme for FLEX 10K devices, and the PS and Passive Serial Asynchronous (PSA) configuration schemes for FLEX 6000.

After compilation, you can also create TTFs that support other configuration schemes for FLEX 6000, FLEX 8000, and FLEX 10K devices.

t_{CO} (Clock to output delay) The time required to obtain a valid output at an output pin that is fed by a register after a Clock signal transition on an input pin that

clocks the register. This time always represents an external pin-to-pin delay.

t_{CO} is also a timing assignment that specifies the maximum acceptable Clock to output delay. In MAX+PLUS II, you can specify a required t_{CO} for an entire project and/or for any input pin (`INPUT` or `INPUTC`), output pin (`OUTPUT` or `OUTPUTC`), or `TRI` buffer (i.e., `BIDIR` or `BIDIRC` pin output function) pin.

TDF *see* Text Design File.

ternary operator An operator that selects between two expressions within an AHDL arithmetic expression. The ternary operator is used in the following format:

<expn 1> ? *<expn 2>* : *<expn 3>*

If the first expression is non-zero (true), the second expression is evaluated and given as the result of the ternary expression. Otherwise, the third expression is evaluated and given as the result of the ternary expression.

Text Design Export File (.tdx) An ASCII text file (with the extension **.tdx**) in AHDL that is optionally generated when you compile a Xilinx Netlist Format File (**.xnf**). It contains the same logic as the XNF File.

A Text Design Export File can be saved as a Text Design File (**.tdf**) and used to replace the corresponding XNF File in the hierarchy of a project.

Text Design File (.tdf) An ASCII text file (with the extension **.tdf**) written in AHDL. Text Design Export Files (**.tdx**) and Text Design Output Files (**.tdo**) can be saved as TDFs and compiled with MAX+PLUS II.

Text Design Output File (.tdo) An ASCII text file (with the extension **.tdo**), generated by the MAX+PLUS II Compiler, that contains the AHDL equivalent of the fully optimized logic for a device in the project.

The Compiler generates a TDO File, as well as an Assignment & Configuration Output File (**.aco**) when you compile a project if you turn on the **Generate AHDL TDO File** command (Processing menu).

You can save a TDO File as a Text Design File (**.tdf**) and recompile it. (You must also save the Assignment & Configuration Output File (**.aco**) as an Assignment & Configuration File (**.acf**) if you wish to preserve the assignments for the device.) TDO Files facilitate back-annotation and preserve the existing logic synthesis in the project.

time unit A unit for specifying a time in MAX+PLUS II. Five time units are available:

Time Unit:	Abbreviation for:
ns	nanosecond
ms	millisecond
us	microsecond
s	second
mhz	megahertz

A time value is always followed immediately (i.e., with no space in between) by a time unit. If you do not enter a time unit, either ns or mhz is assumed as the default, depending on the appropriate context.

Timing Analyzer Output File (.tao) An ASCII text file (with the extension **.tao**) that is used to save the results of the current

timing analysis displayed in the MAX+PLUS II Timing Analyzer.

timing assignment An assignment that specifies desired speed performance on one or more logic functions.

The t_{PD}, t_{SU}, t_{CO}, and f_{MAX} timing assignments, as well as "timing cuts," are available. You can assign timing to individual logic functions and specify default timing for the project as a whole. Timing assignments influence project compilation only for the FLEX 6000, FLEX 8000, and FLEX 10K device families.

timing cut A type of timing assignment that cuts the connections between the timing path for an individual node and other nodes in the project, to indicate that the Compiler should not consider the delay along this path when it attempts to meet the user's desired speed performance while processing the project.

t_{PD} (input to non-registered output delay) The time required for a signal from an input pin to propagate through combinatorial logic and appear at an external output pin.

t_{PD} is also a timing assignment that specifies the maximum acceptable input to non-registered output delay. In MAX+PLUS II, you can specify a required t_{PD} for an entire project and/or for any input pin (`INPUT` or `INPUTC`), output pin (`OUTPUT` or `OUTPUTC`), or `TRI` buffer (i.e., `BIDIR` or `BIDIRC` pin output function) pin.

tri-state buffer A buffer with an input, output, and controlling Output Enable signal. If the Output Enable input is high, the output signal equals the input. If the Output Enable input is low, the output signal is in a state of high impedance. The tri-state buffer is implemented with the `TRI` primitive.

Tri-state buses can be implemented by tying multiple nodes together in a Graphic Editor file and with the `TRI_STATE_NODE` variable in an AHDL file.

t_{SU} (Clock setup time) The length of time for which data that feeds a register via its data or Enable input(s) must be present at an input pin before the Clock signal that clocks the register is asserted at the Clock pin.

t_{SU} is also a timing assignment that specifies the maximum acceptable Clock setup time. In MAX+PLUS II, you can specify a required t_{SU} for an entire project and/or for any input pin (`INPUT` or `INPUTC`) or bidirectional pin (`BIDIR` or `BIDIRC` input function).

TTF *see* Tabular Text File.

Turbo Bit and **logic cell Turbo Bit** A control bit for choosing speed and power characteristics of an Altera device. The *Turbo Bit* logic option is most effective when applied to megafunctions, macrofunctions, and pins. If the Turbo Bit is on, the speed increases; if it is off, the power consumption decreases. The Turbo Bit can be turned on or off in a design file or the Compiler.

Turbo Bit availability differs for each device family

Altera Device Family:	Turbo Bit Availability:
Classic	Applies to the entire device (specified as a device option)
MAX 5000	Not available
MAX 7000	Applies to individual logic cells within a device (specified as a logic option)
MAX 9000	Applies to individual logic cells within a device (specified as a logic option)
FLEX 6000	Not available
FLEX 8000	Not available
FLEX 10K	Not available

In MAX 7000E devices, the *Turbo Bit* option and the *Slow Slew Rate* option on output pins are controlled by a single bit. Therefore, only one of these options can be turned on on an output pin at any one time. If both options are turned on or if both options are turned off, the Compiler uses the *Slow Slew Rate* setting and ignores the *Turbo Bit* setting. On input pins and buried logic cells, the Compiler uses the *Turbo Bit* setting and ignores the *Slow Slew Rate* setting.

two's complement A system of representing binary numbers in which the negative of a number is equal to its inverse plus 1. Arithmetic operators in AHDL assume that groups they operate on are a two's complement binary number. In VHDL, you must declare a two's complement binary number with a signed data type.

U

user libraries One or more directories that contain your own megafunctions, macrofunctions, Symbol Files (**.sym**), AHDL Include Files (**.inc**), or precompiled, user-defined VHDL packages.

The Compiler automatically searches for these user-specified libraries when it compiles a project. The Compiler's **VHDL Netlist Settings** command (Interfaces menu) specifies VHDL design libraries for the current project. You can specify which directories contain your other user libraries with the **User Libraries** command (Options menu) in any MAX+PLUS II application.

V

variable A name that represents a node. In AHDL, a variable can also represent a state machine or an instance of a primitive, megafunction, or macrofunction and is declared in the Variable Section. In VHDL, variables have a single current value, and are declared and used only in processes and subprograms. A VHDL variable is declared with a Variable Declaration; the value of a variable can be modified with a Variable Assignment Statement.

VCC A high-level input voltage represented as a high (1) logic level in binary group values.

In an AHDL Text Design File (**.tdf**), `VCC` is a predefined constant and keyword, and the default active node value. In a VHDL Design File (**.vhd**), `VCC` is represented by `'1'`. In a Verilog Design File (**.v**), `VCC` is represented by `1`. In a Graphic Editor file, `VCC` is a primitive symbol. `VCC` is represented as a high (1) logic level in the Simulator and Waveform Editor.

vector A vector specifies the logic levels for an individual node within a project. The

Simulator uses vectors to simulate the behavior of the project; the Programmer and Simulator use vectors for functional testing.

Vectors for simulation and functional can be defined in Vector Files (**.vec**) or Simulator Channel Files (**.scf**). Functional testing vectors can also be stored in programming files.

Vectors for design entry can be defined in a Waveform Design File (**.wdf**).

Vector File (.vec) An ASCII file (with the extension **.vec**) that contains vectors that specify the logic levels of input nodes in a project. The Simulator uses this file to test the logical operation of the project; the Programmer and Simulator can also use a Vector File for functional testing.

In addition, a Vector File can also be converted into a Waveform Design File (**.wdf**) for design entry.

Verilog Design File (.v) A Verilog HDL File (with the extension **.v**) created with the MAX+PLUS II Text Editor or any other standard text editor. Verilog Design Files can be compiled with the MAX+PLUS II Compiler.

Verilog HDL A Hardware Description Language (HDL).

You can create a Verilog Design File (**.v**) with the MAX+PLUS II Text Editor or any standard text editor and compile it directly with MAX+PLUS II.

You can also generate an EDIF 2 0 0 or 3 0 0 netlist file from a Verilog HDL design that has been processed with a Verilog HDL synthesis tool, then import the file into MAX+PLUS II as an EDIF Input File (**.edf**). In addition, you can directly process a Verilog Design File (**.v**) in MAX+PLUS II with Synopsys tools if you turn on the **Synopsys Compiler** command (Interfaces menu). The MAX+PLUS II Compiler can also generate a Verilog Output File (**.vo**) that contains functional and timing information for simulation with a standard Verilog HDL simulator.

Verilog Output File (.vo) A Verilog Hardware Description Language (HDL) standard netlist file (with the extension **.vo**) that is generated by the Verilog Netlist Writer module of the Compiler. This file can be exported to an industry-standard Verilog HDL simulator for simulation. A Verilog Output File cannot be compiled with the MAX+PLUS II Compiler.

VHDL Very High Speed Integrated Circuit (VHSIC) Hardware Description Language.

You can create a VHDL Design File (**.vhd**) with the MAX+PLUS II Text Editor or any standard text editor and compile it directly with MAX+PLUS II. You can also generate an EDIF 2 0 0 or 3 0 0 netlist file from a VHDL design that has been processed with a VHDL synthesis tool, then import the file into MAX+PLUS II as an EDIF Input File (**.edf**). The MAX+PLUS II Compiler can also generate a VHDL Output File (**.vho**) that contains functional and timing information for simulation with a standard VHDL simulator, and a VHDL Memory Model Output File (**.vmo**) that contains simulation models for a RAM or ROM block.

VHDL Design File (.vhd) An ASCII text file (with the extension **.vhd**) written in VHDL. VHDL Design Files can be compiled by the MAX+PLUS II Compiler.

VHDL Memory Model Output File (.vmo) A Compiler-generated VHDL standard netlist file (with the extension **.vmo**) that contains VHDL simulation models. The Compiler automatically generates a VHDL Memory Model Output File for the project when it generates an EDIF Output File (**.edo**) that contains one or more RAM or ROM blocks.

VHDL Output File (.vho) A VHDL standard netlist file (with the extension **.vho**) that is generated by the VHDL Netlist Writer module of the Compiler. This file can be exported to an industry-standard VHDL simulator for simulation. A VHDL Output File cannot be compiled with the MAX+PLUS II Compiler.

VITAL (VHDL Initiative Toward ASIC Libraries) An industry-standard format for VHDL simulation libraries.

MAX+PLUS II provides VHDL cell simulation models that are compliant with VITAL version 2.2b and version 3.0 (VITAL 95). In addition, the EDIF, VHDL, and Verilog Netlist Writer modules of the MAX+PLUS II Compiler can generate an optional Standard Delay Format (SDF) Output File (**.sdo**) for use with VITAL-compliant simulation libraries.

VMO File *see* VHDL Memory Model Output File.

W

Waveform Design File (.wdf) A graphical waveform design file (with the extension **.wdf**) created with the MAX+PLUS II Waveform Editor. This file represents logic with high (1), low (0), undefined (X), and high-impedance (Z) waveforms. It can also include state machines with waveforms that represent different state names.

waveform interval In the Waveform Editor, an interval is a segment of a node or group waveform that represents its logic level or state name over a specific period of time.

WDF *see* Waveform Design File.

X

Xilinx Netlist Format (XNF) File (.xnf) A netlist file (with the extension **.xnf**) generated by Xilinx software. XNF Files that are generated by running the Xilinx LCA2XNF utility can be compiled directly by the MAX+PLUS II Compiler. An XNF File can define all logic in a project, or be incorporated at the bottom level in a hierarchical project.

Index

Nonalphabetic

Numerics

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W

X

Z