

A+ plus **User Guide**

Copyright © 1985, 1986, 1987
Altera Corporation

A+PLUS User Guide
Version 5.0
September 1987

P25-02218-00

Changes are made periodically to the information contained in this manual. These changes will be incorporated into subsequent editions.

Altera Corporation
3525 Monroe Street
Santa Clara, CA 95051
(408) 984-2800
TELEX: 888496

Copyright © 1985, 1986, 1987 Altera Corporation. All rights reserved.

No part of this manual may be copied or reproduced in any form or by any means without the prior written permission of Altera Corporation.

A+PLUS, SAM+PLUS, LogicMap, Turbo-Bit, MacroMuncher, SAM, BUSTER, EP310, EP320, EP600, EP610, EP900, EP910, EP1210, EP1800, EPB1400, EPS444, and EPS448 are trademarks of Altera Corporation. LogiCaps is a registered trademark of Altera. WordStar is a registered trademark of MicroPro Corporation. Fido is a trademark of Tom Jennings. MS-DOS is a trademark of Microsoft Corporation. FutureNet DASH is a trademark of FutureNet Corporation. PC-CAPS and PC-LOGS are trademarks of Personal CAD Systems, Inc. IBM Personal Computer is a registered trademark of International Business Machines Corporation.

Read This First...

A+PLUS software documentation consists of two manuals:

- ***A+PLUS User Guide***
- ***A+PLUS Reference Guide***

The ***A+PLUS User Guide*** provides installation and operation information for the Altera Programmable Logic User System (A+PLUS™). It is divided into three major parts: **Introduction to A+PLUS**, **Design Entry**, and **Simulation**. If you are a first-time user, you are advised to read Sections 1, 2, and 3 in the **Introduction to A+PLUS** and then—depending on your choice of design entry method—the appropriate section in **Design Entry**. Some of the design entry packages are optional; therefore, they may not be part of your version of the manual. Each of the design entry sections contains a sample session(s) that guides you through the complete process of programming an EPLD with A+PLUS. **Simulation** contains the optional *Functional Simulator*. Following Sections 1 through 3, you will find an *Index* covering the information in the ***A+PLUS User Guide***, ***A+PLUS Reference Guide***, and the optional packages *State Machine Entry* and *Functional Simulator*.

The ***A+PLUS Reference Guide*** contains detailed information on Altera primitives, the Altera Design File format, APLUS and ADP Menus, Utilization Reports, and A+PLUS messages. Also included are several *Appendixes*, a *Glossary*, and an *Index* covering the information in the ***A+PLUS User Guide***, ***A+PLUS Reference Guide***, *State Machine Entry*, and *Functional Simulator*.

At the back of each manual, you will find a Customer Comment Form, a Problem Report Card, and a Warranty Card.



A+PLUS hardware and EPLD programming are described in the ***LogicMap II*** manual.

The **A+PLUS User Guide** contains the following sections:

I. Introduction to A+PLUS

Functional Description Gives an overview of A+PLUS that helps you to determine which design entry method is most appropriate for your particular application, and provides detailed information on the Altera Design Processor.

Installation Lists system components and provides specific instructions for installing A+PLUS software.

Getting Started Gives a brief introduction to the menus and menu functions used in running A+PLUS software.

II. Design Entry

Boolean Equation Entry Describes how to enter and process a design exclusively with Boolean equations.

State Machine Entry (Option) Describes how to enter and process a design with state machines.

Schematic Capture With FutureNet (Option) Describes how to enter a schematic design with FutureNet's Schematic Designer and process it with A+PLUS.



Altera's schematic capture package, LogiCaps, is described in the **LogiCaps** manual.

III. Simulation

Functional Simulator (Option) Explains how to simulate a design with Altera's Functional Simulator.

Index

The ***A+PLUS Reference Guide*** contains the following sections:

Altera Primitive Library Describes each Altera primitive in detail, including block diagrams.

Altera Design File Format Explains the format of the Altera Design File.

A+PLUS and ADP Reference Provides detailed descriptions of the APLUS Menu; the user-controllable and automatic functions of the ADP; and DOS command line processing techniques.

Utilization Report Explains the Utilization Report and gives sample reports for various Altera EPLDs.

A+PLUS Messages Lists all error, information, and warning messages generated by A+PLUS and the ADP, their causes, and suggestions for corrective action.

Appendix A *Part-Specific Information*

Appendix B *Altera Design File-to-Logic Equation File Translation*

Appendix C *BNF Rules*

Appendix D *JEDEC Standard File Format*

Appendix E *Electronic Design Support Service*

Appendix F *Altera Primitive Library (Foldouts)*

Glossary

Index

Manual Updates

Altera documentation is updated with Change Pages, Section Reprints, and a **READ.ME** file.


Change Pages are issued for minor changes to the manual. New information is identified with vertical change bars in the margins next to the changed text. In addition, the date of issue is printed at the bottom of each page.

Section Reprints are issued if a section requires a substantial number of changes. The date of issue is indicated at the bottom of each page.

A **READ.ME File** is provided on the **A+PLUS INSTALL** diskette. This file contains information about recent changes to the A+PLUS software that are not yet reflected in the manual.

Printing Conventions

The following notational conventions are used throughout this manual:

- | | |
|---|--|
| Times Bold | <ul style="list-style-type: none">– A+PLUS commands, prompts, and messages– All types of user input– Primitive names– Key names (enclosed in < >'s) |
| Times Light | <ul style="list-style-type: none">– Most screen and file output |
| <i>Helvetica Bold Italics</i> | <ul style="list-style-type: none">– References to Altera manual titles |
| <i>Helvetica Light Italics</i> | <ul style="list-style-type: none">– References to sections within Altera manuals |
|  | <ul style="list-style-type: none">– Indicates information that requires special attention |

Contents

Read This First	iii
Manual Updates	vii
Printing Conventions	viii

I. Introduction to A+PLUS

Section 1: Functional Description

Overview.....	1-2
Using A+PLUS Software.....	1-5
Design Entry	1-6
Schematic Design Entry	1-6
LogiCaps	1-6
FutureNet (DASH)	1-7
State Machine Entry.....	1-7
Boolean Equation Entry.....	1-8
Netlist Entry.....	1-8

Section 1: Functional Description (Continued)

The Altera Design Processor	1-9
Flattener Module	1-9
Translator Module	1-9
Expander Module	1-10
Logic Equation Files	1-10
Minimizer Module.....	1-11
LEF Analyzer Module.....	1-11
Fitter Module.....	1-11
Assembler Module.....	1-12
Functional Simulator	1-13
LogicMap II.....	1-14

Section 2: Installation

Stop — Read this first.....	2-2
A+PLUS Distribution Diskettes	2-3
Backups of Altera Distribution Diskettes	2-8
A+PLUS Installation.....	2-9
De-Installation.....	2-13

Section 3: Getting Started

APLUS Menu	3-2
ADP Menu	3-4

II. Design Entry

Boolean Equation Entry

Who Should Use Boolean Equation Entry?	BE-1
General Requirements	BE-2
Functional Description	BE-3
Sample Session	BE-5
The Sample Circuit.....	BE-5
Entering the Sample Design.....	BE-8
Altera Design File (ADF) Format	BE-15
Header Section Requirements	BE-17

Boolean Equation Entry (Continued)

Declarations Section Requirements	BE-17
Options Section	BE-18
Part Section	BE-18
Inputs Section	BE-19
Outputs Section	BE-20
Network Section Requirements	BE-20
Naming Conventions	BE-27
Inputs	BE-27
I/O (No Feedback)	BE-27
I/O (With Feedback)	BE-27
I/O (No Output)	BE-28
Bus I/O Primitives	BE-28
Active Low Signals	BE-31
Equations Section Requirements	BE-32
End Statement	BE-34
Additional Guidelines	BE-35

(Tables of contents for optional packages are included in each individual package.)

III. Simulation

(Tables of contents for optional packages are included in each individual package.)

Index

Illustrations

Figure		Page
1-1.	A+PLUS Block Diagram.....	1-4
2-1.	Installation Main Menu.....	2-10
2-2.	Installation Menu.....	2-11
2-3.	De-Installation Menu	2-13
BE-1.	Boolean Equation Entry.....	BE-4
BE-2.	Sample Circuit	BE-4
BE-3.	ADF for the Sample Design (BEVDIS)	BE-11
BE-4.	Sample ADF	BE-16
BE-5.	Sample ADF for EP1210.....	BE-37
BE-6.	Sample ADF for EP600.....	BE-38
BE-7.	Sample ADF for EP320.....	BE-39

Tables

Table		Page
BE-1.	Input, I/O, and Bus I/O Syntax.....	BE-24
BE-2.	Recommended Naming Conventions	BE-29
BE-3.	ADF Legal Pin Name and Node Name Characters.....	BE-30

SECTION 1

Functional Description

This section gives an overview of the Altera Programmable Logic User System (A+PLUS), followed by a detailed description of each component of the A+PLUS software.

Overview

The Altera Programmable Logic User System (A+PLUS) is a combination of software and hardware that allows circuit designers to develop and implement custom logic circuits with Altera EPLDs. A+PLUS provides an ideal desktop environment for EPLD design entry, compilation, fitting, and programming.

The system contains A+PLUS software, a powerful design support tool developed to meet the diverse needs of design engineers and to match the continually expanding capabilities of Altera EPLDs. A+PLUS hardware consists of a software-controlled Logic Programmer card and an external Logic Programmer unit used for device programming.

Figure 1-1 shows a block diagram of A+PLUS.

A+PLUS software allows you to create a design with schematic capture, state machine, Boolean equation, or netlist entry methods. Your design file is converted (if necessary) to the Altera Design File (ADF) format, and then processed into an industry-standard JEDEC file by the Altera Design Processor (ADP). Finally, the LogicMap II program translates the JEDEC file into a working part via the Logic Programmer unit.

A+PLUS software also provides the following features:

- Supports Altera EPLDs.
- Supports automatic part selection.
- Supports Altera's Standard and TTL MacroFunction libraries.
- Supports user-defined MacroFunctions created with ADLIB.
- Applies sophisticated logic reduction algorithms to ensure optimum resource utilization in your design.
- Automatically generates a Utilization Report showing how your design will be implemented in the programmable logic device.
- Supports functional simulation with Altera's Functional Simulator.
- Translates a design into a working part with a fully automated integration process, that makes pin assignments optional.

A+PLUS software runs on an IBM PC XT, PC AT, or other compatible computers capable of running MS-DOS Version 2.0 or greater. The computer must have at least one floppy disk drive capable of reading 360-Kbyte, double-sided, double-density disks; a 10-Mbyte hard-disk drive; 640 Kbytes of RAM memory; and one full-length expansion slot for the Altera Logic Programmer card.

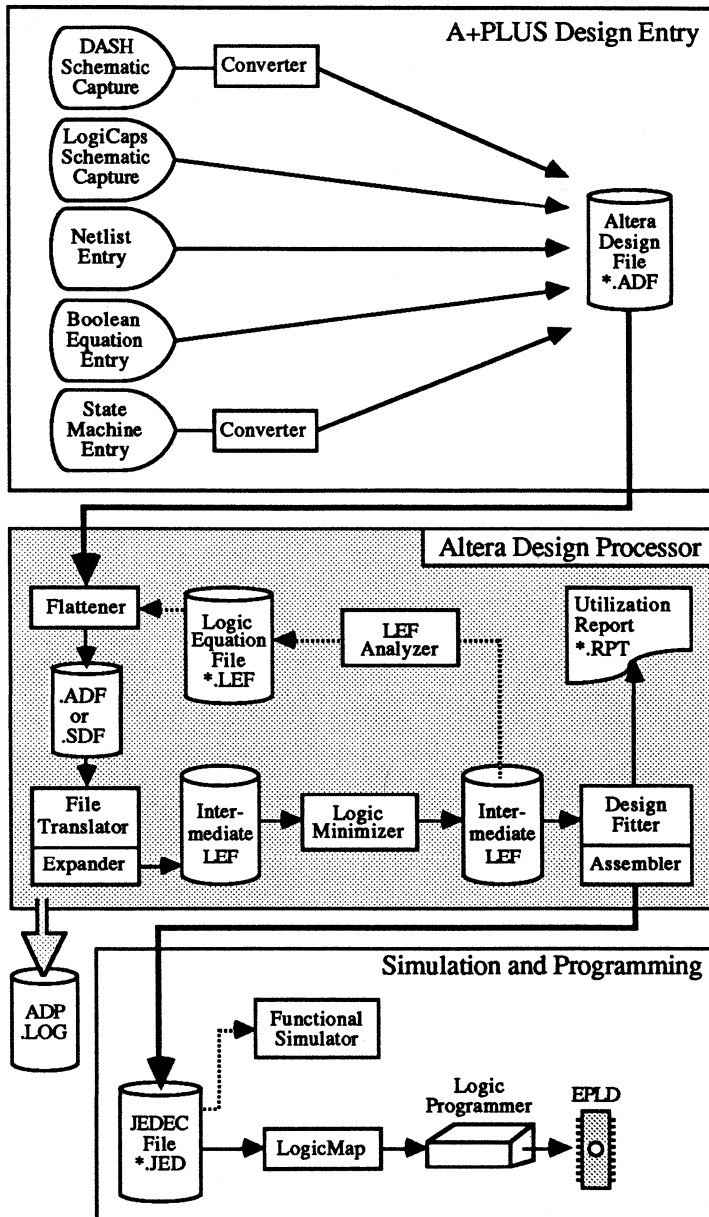


Figure 1-1. A+PLUS Block Diagram

Using A+PLUS Software

Before beginning design entry, you should familiarize yourself with the Altera Primitive Library. This library contains input, basic gate, equation, flipflop I/O, and Bus I/O symbols, which are the functional blocks used in designing circuits with A+PLUS software. All Altera-supplied primitives are described in detail in *Altera Primitive Library* in the ***A+PLUS Reference Guide***.

You should also have a general understanding of the functions of the Altera Design Processor (ADP), whose components are described later in this section. However, since the ADP controls the processing sequence, minimal user interaction is required.

If you are a LogiCaps user, you should also familiarize yourself with the Altera Standard MacroFunctions and the (optional) TTL MacroFunctions. Altera MacroFunctions are high-level building blocks that, when used with Altera primitives, can greatly increase design productivity. The Altera MacroFunctions are described in the ***LogiCaps*** and ***TTL MacroFunctions*** manuals.

Design Entry

A+PLUS software supports four design entry methods: schematic entry, state machine entry, Boolean equation entry, and netlist entry. You may choose whichever method best suits your design entry task. However, you are not restricted to one entry format, but may use any combination of entry methods, submitting multiple files to be compiled into a single EPLD.

Each supported entry method ultimately creates a netlist file, called an Altera Design File (ADF), which is the common entry format for the A+PLUS software. The Altera Design Processor (ADP) then processes the ADF and generates a JEDEC file for programming the EPLD.

Each design entry method is described below.

Schematic Design Entry

The schematic entry method is particularly suited for large random logic functions. The optional graphic schematic capture modules—LogiCaps and DASH—allow you to enter a schematic diagram directly into the computer with the Altera Primitive Library.

A+PLUS software also supports design entry with PC-CAPS (developed by Personal CAD Systems, Inc.) For additional information, refer to *Schematic Capture With P-CAD* (Version 1.3).

LogiCaps

LogiCaps is a high-performance schematic capture package that has been optimized for entering designs destined for Altera EPLDs. It allows you to create a design with any symbol available in the Altera Primitive Library and the Standard and TTL MacroFunction Libraries. LogiCaps directly outputs an Altera Design File (with the extension .ADF), which the Altera Design Processor uses to process the design and generate a JEDEC file for programming the EPLD.

LogiCaps offers a multitude of features, including dual editing windows, multiple zoom levels, and printer interface software. (For additional information, refer to the *LogiCaps* manual.)

FutureNet (DASH)

The DASH Schematic Designer is a hardware and software package developed by FutureNet Corporation. DASH software allows you to enter new designs, including arbitrary Boolean expressions, with Altera Primitive Library symbols. It also enables you to edit existing designs and make hard-copy printouts of your design

The DASH program outputs a pinlist that represents your design. When the Altera Design Processor is invoked, the A+PLUS FutureNet Pin List Converter automatically translates this list into an ADF. (For further information, see *Schematic Capture With FutureNet*.)

State Machine Entry

The optional state machine entry method is suitable for users familiar with describing the logical operation of state machine designs with Boolean expressions, truth tables, state diagrams, and Algorithmic State Machine charts.

Using any standard text editor, you generate a State Machine File (SMF). Your SMF may contain Boolean and non-Boolean primitive statements, Boolean equations, and truth tables. The A+PLUS State Machine Converter (SMV) automatically translates this file into an Altera Design File when the Altera Design Processor is invoked. (For further information, see *State Machine Entry*.)

Boolean Equation Entry

Boolean Equation Entry is the traditional entry method for programmable logic, and is appropriate for designing with low-density EPLDs. The A+PLUS software supports Boolean equation entry directly via the Altera Design File (ADF) format.

If you choose this entry method, you simply use a standard text editor to type the Boolean equations into the Altera Design File format. The final ADF is submitted to the Altera Design Processor, which processes the design and generates a JEDEC file for programming the EPLD. (For additional information, refer to *Boolean Equation Entry*.)

Netlist Entry

The A+PLUS software also supports netlist entry directly via the Altera Design File (ADF) format.

Using a standard text editor, you can type a netlist corresponding to your design into the ADF format. This entry method also permits circuit designers to utilize netlist output (e.g., from workstations or schematic capture packages that do not support the ADF netlist standard) that has been adapted to the ADF format. The final ADF is submitted to the Altera Design Processor, which processes the design and generates a JEDEC file for programming the EPLD. (For further information on requirements for ADF entry and the ADF format, refer to *Boolean Equation Entry* in the **A+PLUS User Guide** and *Altera Design File Format* in the **A+PLUS Reference Guide**.)

The Altera Design Processor

Using Altera Design Files (ADFs) created with the schematic capture, state machine, Boolean equation or netlist interfaces as input, the Altera Design Processor (ADP) runs a series of modules whose final output is a JEDEC file used for programming an Altera EPLD. (A+PLUS converts design files created with the State Machine and DASH entry methods into standard ADF format before design processing begins.) This process is nearly fully automatic and requires minimal input from the circuit designer.

During processing, the ADP sends error, information, and warning messages both to the computer screen and an **ADP.LOG** file on disk. (See *A+PLUS Messages* in the ***A+PLUS Reference Guide***.)

Each component of the ADP is described below.

Flattener Module

The Flattener module checks each ADF submitted to the ADP for MacroFunction statements, which it expands and replaces with primitive statements. The MacroFunctions are thus “flattened” into primitive elements that preserve the electrical connectivity of the original design. The flattened file is then passed to the Translator module with the extension **.SDF**. If no MacroFunctions are found, the file is simply passed to the Translator module with the extension **.ADF**.

Translator Module

The Translator module converts the Flattener module output into a Logic Equation File (LEF), which expresses the design as a series of Boolean equations. The Translator then performs a thorough examination of the logical completeness and consistency of the design, including checks for design connectivity and syntactical errors. Most errors are detected and can be easily corrected at this stage of design processing. If an invalid request, such as a request for a pin

number that does not exist, is detected, the Translator issues an error message and returns control to the ADP.

The Translator module can also perform automatic part selection, based on the logic requirements of the design. With this option, you need not define the EPLD to be programmed. Instead, you specify "AUTO" and the Translator chooses the EPLD most likely to fit your design.

Expander Module

The Expander module expands the Boolean equations into sum-of-products form, checks for evidence of combinatorial feedback, removes redundant factors from product terms, and produces another LEF.

If illegal combinatorial feedback is detected, you are alerted with an error message, and control is returned to the ADP. (For example, a symbol appearing on both the left- and right-hand sides of an equation causes illegal feedback.)

After all equations have been processed successfully, control passes to the Minimizer module.

Logic Equation Files

Logic Equation Files are internal data structures. The first intermediate LEF is output by the Translator and passed to the Expander module; the second LEF is generated by the Expander and input to the Minimizer module; and the third LEF is output by the Minimizer and input to the Fitter module or, optionally, to the LEF Analyzer module. (For an example of ADF-to-LEF translation, see *Appendix B* in the ***A+PLUS Reference Guide***)

Minimizer Module

At your request, the Minimizer module performs a sophisticated Boolean logic reduction on the translated design to maximize utilization of EPLD resources. Its primary reduction method includes generalized consensus and logical absorption algorithms, which quickly and significantly reduce the number of product terms in equations.

For designs using JK or SR flipflops, the Minimizer checks whether a D or T flipflop will provide a more efficient design implementation in each case. D or T flipflops are substituted where appropriate, and the resulting equations are minimized accordingly.

Following the primary reduction scheme, the Minimizer applies De Morgan's inversion theorem. If you wish, you can choose to apply logical inversion on an equation-by-equation basis.

If abnormal termination is caused, for example, by insufficient internal storage, the Minimizer displays an error message and returns control to the ADP.

LEF Analyzer Module

At your request, the Analyzer module converts the intermediate LEF output by the Expander (or, if minimization is requested, the Minimizer) into a readable format similar to that of the ADF, which appears as a file with the extension `.LEF`. You may then check the design for possible problems, such as an excess of product terms. If you find a problem, you can correct the input design file and resubmit it to the ADP.

Fitter Module

The Fitter module begins the final conversion of the Logic Equation File into a JEDEC file format. Using the LEF output by the Minimizer, the Fitter matches the requirements of your design with the known resources of an Altera device. It places each logic function in the best location and selects appropriate interconnection paths and pin assignments. If you wish, you can specify some or all pin assignments,

and the Fitter will attempt to match these requests with the resources on the pins. If it cannot find a fit, it issues an information message and provides you with the option to remove your pin assignments and continue the fitting process.

Regardless of whether a fit is achieved, the fitting process generates a Utilization Report file (with the extension **.RPT**) that documents macrocell and pin assignments, input and output pin names, and buried registers, as well as any unused resources. (For a sample Utilization Report and detailed explanations, see *Utilization Report* in the **A+PLUS Reference Guide**.)

Assembler Module

The Assembler Module completes design processing by converting the Fitter's requests into a programmable image for the part, in the form of a JEDEC file (with the extension **.JED**) that includes the header information from the Altera Design File(s). The JEDEC file is then processed by LogicMap II and the Altera Logic Programmer unit to produce a working device.

Functional Simulator

Altera's optional Functional Simulator package is a time-saving tool for testing the logical operation of an EPLD design. It uses specified design and part information to model the operation of the EPLD before the design is actually committed to hardware. (For additional information, see *Functional Simulator*.)

LogicMap II

LogicMap II is the interface software that programs EPLDs from JEDEC files generated by the Altera Design Processor. Before programming a part, it enables you to view and edit a JEDEC file through a series of hierarchical windows. (Note: JEDEC file editing is not available for the BUSTER EPLD [EPB1400].) The programming process allows you to program, examine, and verify an Altera EPLD with the external Logic Programmer unit. (For further information, refer to the *LogicMap II* manual.)

SECTION 2

Installation

This section assists you with the installation of A+PLUS software.

For instructions on how to install A+PLUS hardware, refer to *Installation* in the ***LogicMap II*** manual.

If you have any questions regarding installation, please contact:

Altera Corporation
Applications Dept.
3525 Monroe Street
Santa Clara, CA 95051
(408) 984-2805 ext. 102

Stop — Read this first ...

Please be sure to go through the installation procedure in the following order. Refer also to the related hardware installation procedure in the *LogicMap II* manual. For your convenience, blank boxes are provided so you may check off each item (✓) after you have completed it.

- Record the serial numbers of your programming hardware before beginning the installation.
- Fill out the Warranty Registration card. (If you have purchased software only, write 0 for the serial numbers.)
- Read the README file on your INSTALL diskette for changes that may have been made to the installation procedure since the manual was printed.
- Read the installation instructions for installing the hardware in the *LogicMap II* manual.
- Install the programming hardware.
- Read the installation instructions for installing the software in the *A+PLUS User Guide*.
- Install the A+PLUS software.
- Mail the Warranty Registration card to Altera. You will receive all future update information for A+PLUS software *only* if you mail this card.

A+PLUS Distribution Diskettes

A+PLUS software requires DOS version 2.0 or greater. DOS version 3.3 is recommended.

The following list describes all available Altera distribution diskettes. Depending on which development system you have purchased, you may have some or all of these diskettes. Refer to the lists on the following pages to verify which diskettes should accompany your development system.

- **INSTALL**
Contains the installation procedures for all A+PLUS modules.
- **APLUS**
Contains the A+PLUS programs and their support files.
- **ADP**
Contains the Altera Design Processor (ADP).
- **LOGICMAP**
Contains the LogicMap II program and support files.
- **ECF**
Contains data files used by LogicMap II.
- **SMV**
Contains the State Machine Converter and support files.
- **LOGICAPS**
Contains the LogiCaps schematic capture program.
- **UTILITIES**
Contains the LogiCaps utilities.
- **MACROLIB**
Contains the Altera Standard MacroFunctions.
- **MACROLIB-TTL**
Contains the Altera TTL MacroFunctions.
- **ADLIB**
Contains the Altera Design Librarian.

- **FSIM**
Contains the Functional Simulator program.
- **FNET**
Contains the Altera Primitive Library and interface programs for use with FutureNet's DASH Schematic Designer.

If You Have Purchased PLCAD-SUPREME...

...you should have the following distribution diskettes:

- **INSTALL**
Contains the installation procedures for all A+PLUS modules.
- **APLUS**
Contains the A+PLUS programs and their support files.
- **ADP**
Contains the Altera Design Processor (ADP).
- **LOGICMAP**
Contains the LogicMap II program and support files.
- **ECF**
Contains data files used by LogicMap II.
- **SMV**
Contains the State Machine Converter and support files.
- **LOGICAPS**
Contains the LogiCaps schematic capture program.
- **UTILITIES**
Contains the LogiCaps utilities.
- **MACROLIB**
Contains the Altera Standard MacroFunctions.
- **MACROLIB-TTL**
Contains the Altera TTL MacroFunctions.
- **ADLIB**
Contains the Altera Design Librarian.
- **FSIM**
Contains the Functional Simulator program.

If You Have Purchased PLCAD4...

...you should have the following distribution diskettes:

- **INSTALL**
Contains the installation procedures for all A+PLUS modules.
- **APLUS**
Contains the A+PLUS programs and their support files.
- **ADP**
Contains the Altera Design Processor (ADP).
- **LOGICMAP**
Contains the LogicMap II program and support files.
- **ECF**
Contains data files used by LogicMap II.
- **LOGICAPS**
Contains the LogiCaps schematic capture program.
- **UTILITIES**
Contains the LogiCaps utilities.
- **MACROLIB**
Contains the Altera Standard MacroFunctions.
- **MACROLIB-TTL**
Contains the Altera TTL MacroFunctions.
- **ADLIB**
Contains the Altera Design Librarian.

If You Have Purchased PLDS2...

...you should have the following distribution diskettes:

- **INSTALL**
Contains the installation procedures for all A+PLUS modules.
- **APLUS**
Contains the A+PLUS programs and their support files.
- **ADP**
Contains the Altera Design Processor (ADP).
- **LOGICMAP**
Contains the LogicMap II program and support files.
- **ECF**
Contains data files used by LogicMap II.

Backups of Altera Distribution Diskettes

Before installing Altera software, you must make backups of all Altera distribution diskettes in case one of the distribution diskettes fails to work. Note that the copy-protected diskettes are specially formatted. You can copy the files from these diskettes onto a backup diskette, but you cannot start the copy-protected programs with the backup diskette. So, if you lose or damage a program on an original copy-protected diskette, you must copy the backup of the program onto the original copy-protected diskette.



The DOS **DISKCOPY** command cannot be used to duplicate the distribution diskettes. Altera diskettes are copy-protected and **DISKCOPY** is unable to read them.

To make backups of all Altera distribution diskettes, you must go through the following steps:

1. Boot DOS.
2. Format a blank diskette for each distribution diskette with the DOS **FORMAT** command. (Refer to your DOS manual.)
- 3a. If your system has one floppy disk drive and one hard disk drive, insert an Altera distribution diskette into drive **A** and type:

```
COPY A:*. * A: <Enter>
```

You will be prompted to replace the Altera distribution diskette with a formatted diskette.

- 3b. If your system has two floppy disk drives and one hard disk drive, insert the Altera distribution diskette into drive **A** and the formatted diskette into drive **B** and type:

```
COPY A:*. * B: <Enter>
```

4. Repeat step 3 for each Altera distribution diskette.
5. Store your backup diskettes in a safe place.

A+PLUS Installation

After you have completed the following installation procedure, you will be able to run A+PLUS directly from your hard disk.

Before installing A+PLUS software on your hard disk, you must ensure that your computer has at least 1 Mbyte of free disk space on the hard disk and 640 Kbytes of RAM memory, otherwise installation will not be successful. Available space is verified with the DOS **CHKDSK** command. (For information regarding DOS, refer to *IBM Disk Operating System, Version 3.30 User's Guide*; *IBM Disk Operating System, Version 3.30 Reference*; and *IBM DOS Technical Reference*.)



IBM AT high-density 1.2 Mbyte diskette drives are not compatible with the standard 360 Kbyte diskette drives. If you anticipate moving A+PLUS from an AT to an XT computer, you should use a 360 Kbyte diskette drive, if available, to perform the installation.

This procedure assumes that your hard disk is drive **C**. (If you have another hard-disk drive, substitute the appropriate letter.)

1. Boot the computer from the hard disk.



Remove any write-protect tabs that may be on the distribution diskettes.

2. Insert the Altera-provided **INSTALL** diskette into drive **A** or **B** and enter:

A:INSTALL <Enter> (if you use drive **A**)

or

B:INSTALL <Enter> (if you use drive **B**)

The program will perform some basic checks, after which the Installation Main Menu, shown in Figure 2-1, is displayed.

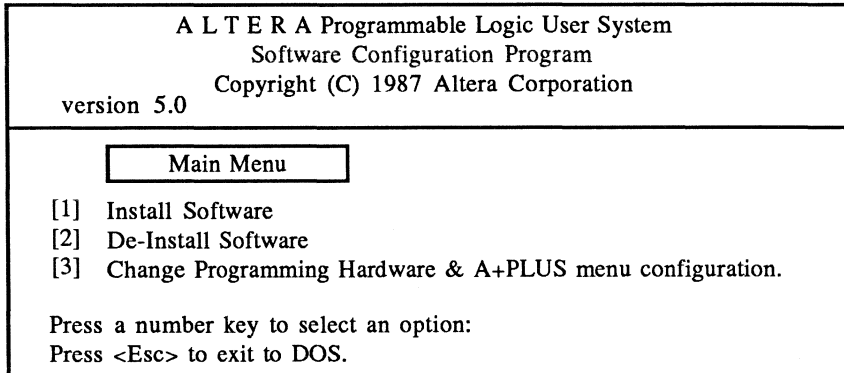


Figure 2-1. Installation Main Menu

- Item [1] Guides you through A+PLUS installation.
- Item [2] De-installs the software from the current system so that you can move the software to another computer.
- Item [3] Allows you to change the A+PLUS hardware installation file called **EPLD.SYS**. This file is used only if you want to change the address location of the programming card or disable the color display option. Refer to the **LogicMap II** manual for valid programming card addresses.

3. Press <1> to select option [1]. A set of prompts will guide you step by step through your system configuration.
4. Next, you are prompted step by step through the actual installation procedure.



This installation process has a master menu that provides installation for all optional Altera products. If you try to install an option which you have not purchased, you will be returned to the Main Menu.

You are guided through the following process:

- a. You create a directory on your hard disk that contains all files from the installation diskette. You are also asked to confirm the installation results.
- b. You are requested to enter the address location of your programming card. Type:

280 <Enter>

This is the default address.

- c. You are asked to indicate whether you wish to use a color or monochrome display.
 - d. You are asked to enter the name of your editor. LogiCaps is the default editor.
5. Now, the Installation Menu is displayed. See Figure 2-2.

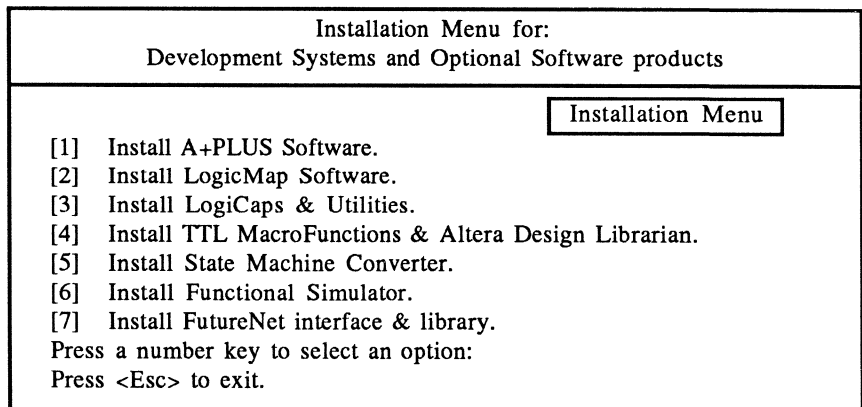


Figure 2-2. Installation Menu

- a. You may install one or more of the programs in any order.
 - b. Once you press a number key, you are prompted to insert the appropriate distribution diskette.
 - c. When you are finished, press <Esc> to return to the Main Menu.
6. You may select other menu items from the Main Menu or press <Esc> to return to DOS.
 7. **Remove any remaining diskette from the floppy disk drive and press <Ctrl><Alt> to reboot the system before using any of the A+PLUS programs.**



Your **CONFIG.SYS** and **AUTOEXEC.BAT** files may have been modified to make LogiCaps run properly. Therefore, after completing the installation procedure, you may want to examine these files to determine whether they are compatible with other software on your system. The original files will have been saved as **CONFIG.BAK** and **AUTOEXEC.BAK**. To run A+PLUS, the **CONFIG.SYS** file *must* contain **BUFFERS = 12** and **FILES = 20**.

De-Installation

If you wish to install your A+PLUS software on a different computer, you must first reverse the installation from your original system.



If you are using a programming card, you must move this card *after* you have de-installed the software and *before* you re-install it on the new computer.

With the original **INSTALL** distribution diskette in drive **A** or **B**, type:

A: INSTALL <Enter>

or

B: INSTALL <Enter>

The Installation Main Menu is displayed (see Figure 2-1).

Select menu item [2] on the Main menu. The De-Installation Menu is displayed, as shown in Figure 2-3.

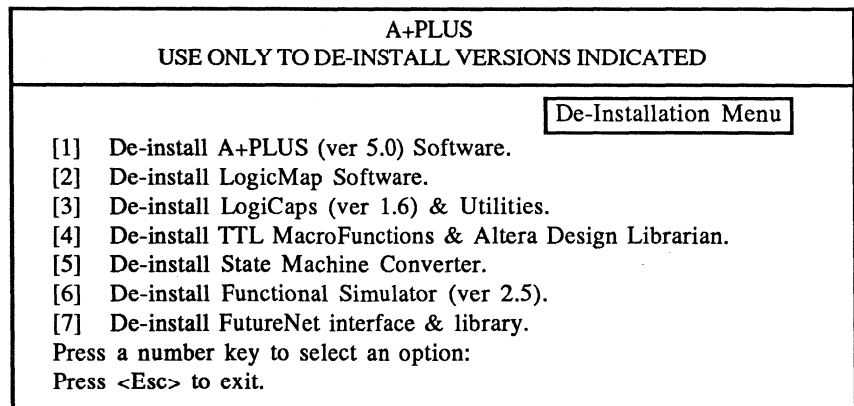


Figure 2-3. De-Installation Menu

Select the products you wish to de-install.

As a result of this de-installation, you can no longer use the A+PLUS software on the hard disk of the first system.

After the initial installation, you cannot use the original distribution diskettes for additional installations unless the de-installation program is run first.



After de-installation, some A+PLUS files that are not application files will be removed from the directory on your hard disk.

SECTION 3

Getting Started

This section helps you get started with A+PLUS as quickly as possible. It provides a brief description of each of the functions available on the APLUS and ADP menus, and shows how the menus appear on your computer screen.

After installing the A+PLUS software and hardware, you can use these menu descriptions to quickly familiarize yourself with the steps used to process a design file into a JEDEC file for programming an Altera EPLD. For comprehensive information on using A+PLUS, including detailed sample sessions, refer to the individual design entry method sections in the ***A+PLUS User Guide***. For more detailed descriptions of the APLUS and ADP menu functions, see *A+PLUS and ADP Reference* in the ***A+PLUS Reference Guide***.

APLUS Menu

To invoke the APLUS Menu, type at the DOS prompt:

APLUS <Enter>

The APLUS Menu is displayed, as shown in Figure 3-1.

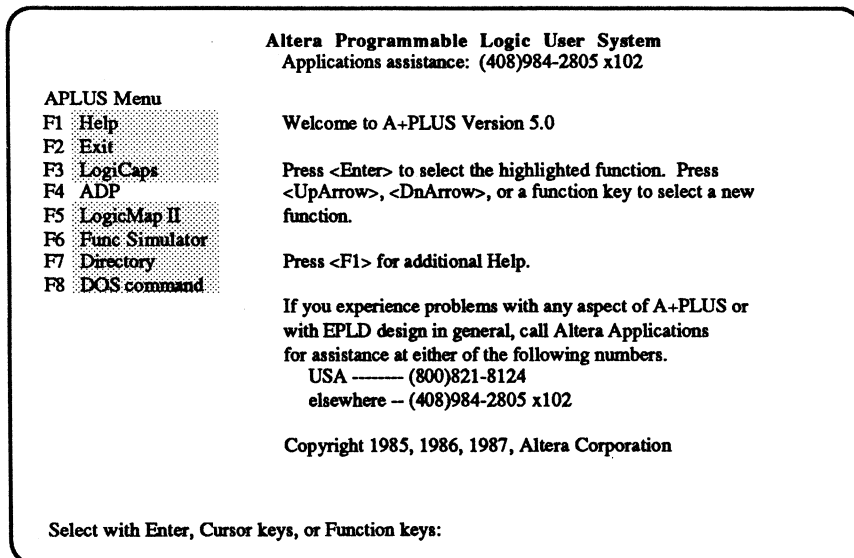


Figure 3-1. APLUS Menu

The current (default) menu selection, <F4> (ADP), is highlighted. The numbers displayed on the left of the menu selections show the function keys used to execute each function. You may also select any function with the cursor keys and <Enter>.

By pressing <F4>, you invoke the menu for the Altera Design Processor. The functions controlled by the ADP Menu allow you to process your design file into a JEDEC file used for programming an Altera EPLD. The ADP Menu functions are described in *ADP Menu* (below).

The other APLUS Menu functions are:

- <F1> **Help** Displays helpful information on APLUS Menu functions. After pressing <F1>, you may use the function or cursor keys to display a help message for each menu function.
- <F2> **Exit** Terminates A+PLUS and returns you to DOS. (Use the **DOS Command** (<F8>) function to execute DOS commands during the current A+PLUS session.)
- <F3> **<LogiCaps>** Invokes the text editor or schematic capture program of your choice. LogiCaps is the default, but you can override the default by entering another editor (e.g., WordStar) during installation.
- <F5> **LogicMap II** Invokes the LogicMap II program, which allows you to program an Altera EPLD with A+PLUS hardware and a JEDEC file generated by the ADP.
- <F6> **Func Simulator** Invokes Altera's Functional Simulator, which allows you to test the logical operation of your design before it is actually committed to hardware.
- <F7> **Directory** Allows you to enter a search pattern and then displays a list of DOS files matching the pattern.
- <F8> **DOS command** Temporarily returns you to DOS so that you can enter a DOS command. (After entering the DOS command, type EXIT to leave the temporary DOS environment and return to A+PLUS.)

ADP Menu

The ADP Menu functions allow you to specify the conditions used for compiling your design file. To invoke the ADP Menu, press <F4> while in the APLUS Menu. The ADP Menu, shown in Figure 3-2, is displayed.

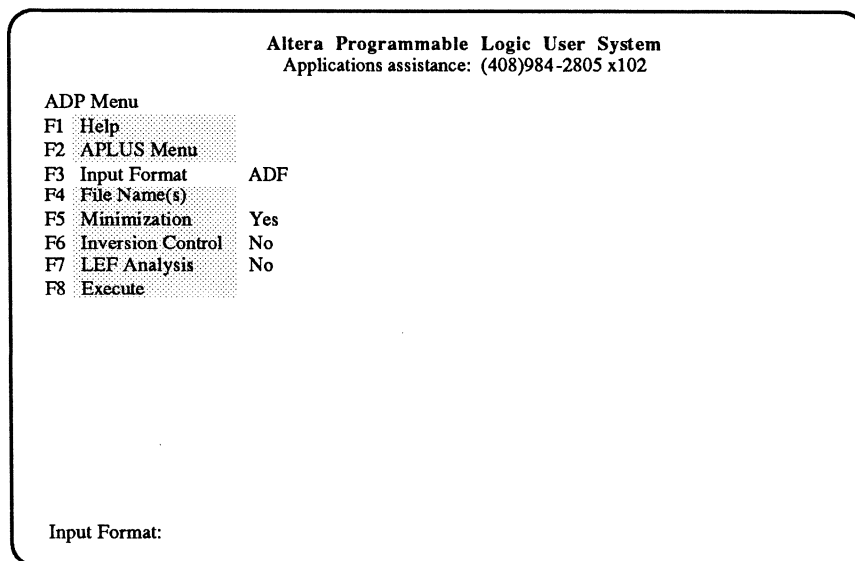


Figure 3-2. ADP Menu

The current (default) menu selection, <F3> (Input Format), is highlighted. This function is the first in the default sequence for entering processing parameters, but you may move around the menu and enter and change processing options in any order. The default processing settings are shown on the right of the function names.

The ADP Menu functions are:

- <F1> **Help** Displays helpful information on ADP Menu functions.
- <F2> **APLUS Menu** Terminates the current ADP session and returns you to the APLUS menu.
- <F3> **Input Format** Prompts you for the format of your input file:
 - A or <Enter> – for an Altera Design File (the default)
 - C – for a component list as output by P-CAD's PC-CAPS
 - P – for a pinlist as output by FutureNet's DASH Schematic Design Editor
 - S – for a State Machine File
- <F4> **File Name(s)** Prompts you for the name(s) of file(s) containing your design. You need not enter the filename extension.
- <F5> **Minimization** Allows you to request a reduction of the Boolean logic of the design. If you enter Y (Yes), Boolean minimization is performed on the design. If you enter N (No), the ADP Menu selection automatically moves to the **LEF Analysis (<F7>)** prompt.
- <F6> **Inversion Control** Allows you to request control over inversion of individual Boolean equations during the minimization process. If you enter N (No—the default), the ADP automatically decides whether or not to invert each equation and retains the form that contains the fewest product terms. If you enter Y (Yes), you are asked on an equation-by-equation basis whether you wish to perform a De Morgan's inversion. This feature is useful mainly for unusual EP310 and EP1210 designs.
- <F7> **LEF Analysis** Allows you to request the ADP to analyze the intermediate Logic Equation File (LEF). The LEF Analyzer creates a file (in a format similar to the Altera Design File format) that allows you to examine the design after it has been translated, expanded, and, if requested, minimized.

After you have answered the ADP Menu prompts, you are asked:

Do you wish to run under the above conditions [Y/N]?

Simply enter **Y**, or press **<F8>** (**Execute**) to execute the ADP. That's all you need to do—the ADP will immediately process the design according to the current menu options and generate a JEDEC file for programming an Altera EPLD.

Boolean Equation Entry

This section describes the process of entering circuit designs with Boolean equations. It includes a list of requirements for designing exclusively with Boolean equations; an extensive sample session; several sample ADFs; a detailed description of the Altera Design File (ADF) format used to enter circuit designs with Boolean equations; and additional design guidelines.

Who Should Use Boolean Equation Entry?

This design entry method is convenient for design engineers who are familiar with different types of programmable logic devices. If you have previous experience with Boolean assemblers, you will find the A+PLUS source code syntax easy to master. The programmable architecture of the Altera EPLDs is controlled in the Network Section of the source code, which is the only section whose syntax will be new to you. Otherwise, you use standard logical operators. This feature allows you to incorporate design logic generated for other devices into your EPLD designs.

General Requirements

To enter your design with Boolean equations, you need:

- A text editor that uses standard ASCII character conventions. (If your word processor has both document and non-document modes, use only the non-document mode.)
- A filename with the extension **.ADF**.

Functional Description

To implement a design exclusively with Boolean equations, you use a standard ASCII text editor to create an Altera Design File (ADF) that describes the circuit. The completed file is submitted to the Altera Design Processor (ADP) which, in turn, produces a JEDEC file. The LogicMap II module then uses the JEDEC file to program an EPLD. Figure BE-1 illustrates this process:

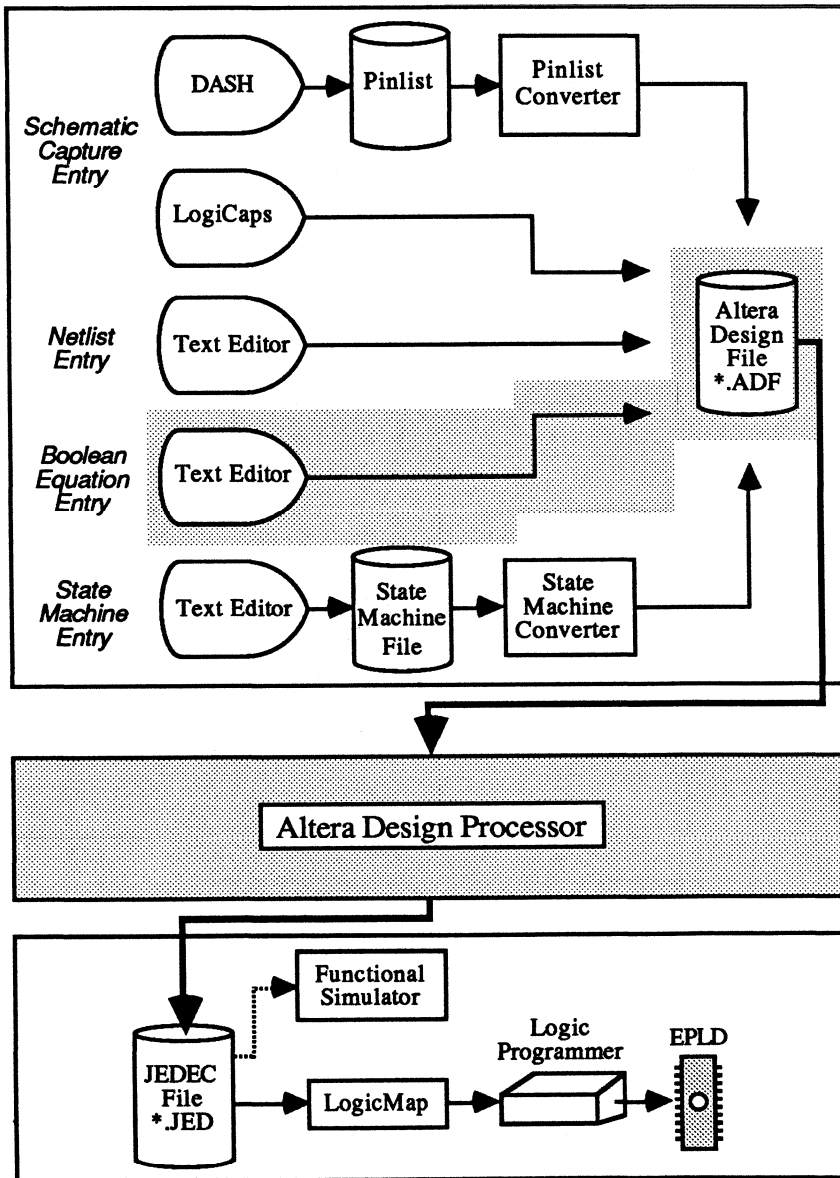


Figure BE-1. Boolean Equation Entry

Sample Session

This sample session provides a step-by-step example of how to enter a design created with Boolean equations and produce a working EPLD. It includes a description of a sample design—a beverage dispenser controller—and instructions for each phase of design entry and processing. We strongly recommend that you go through this sample session before implementing your own design. (For your reference, four additional ADFs are included in the section following this sample session.)

The Sample Circuit

The sample circuit controls the operation of a simplified coin-activated beverage vending machine. Figure BE-2 shows the circuit schematic.

The mechanism has five inputs and three outputs. The **RESET**, **COINDROP**, and **CUPFULL** inputs control the operating conditions of the three outputs of the circuit: **DROPCUP-POURDRNK-STROBE**. (The **STROBE** signal is output for use by other circuits in the machine.)

Using the recommended Altera naming conventions (described in *Network Section Requirements*), the equations describing the behavior of the sample circuit may be expressed as follows:

```
DROPCUPd = COINDROP * /DROPCUP * /POURDRNK;
POURDRKd = DROPCUP * /POURDRNK
          + /CUPFULL * /DROPCUP * POURDRNK;
STROBE = /CLOCK * ((DROPCUP * /POURDRNK)
                  + (/DROPCUP * POURDRNK));
NEWCYCLE = DROPCUP * POURDRNK;
```

Since entire equations may be substituted into the right-hand sides of other equations by substituting intermediate variables, it is also possible to express the behavior of the sample circuit with:

```
DROPCUPd = COINDROP * /DROPCUP * /POURDRNK;  
CUPREADY = DROPCUP * /POURDRNK;  
POURDRKd = CUPREADY  
          + /CUPFULL * /DROPCUP * POURDRNK;  
STROBE = /CLOCK * (CUPREADY  
          + (/DROPCUP * POURDRNK));  
NEWCYCLE = DROPCUP * POURDRNK;
```

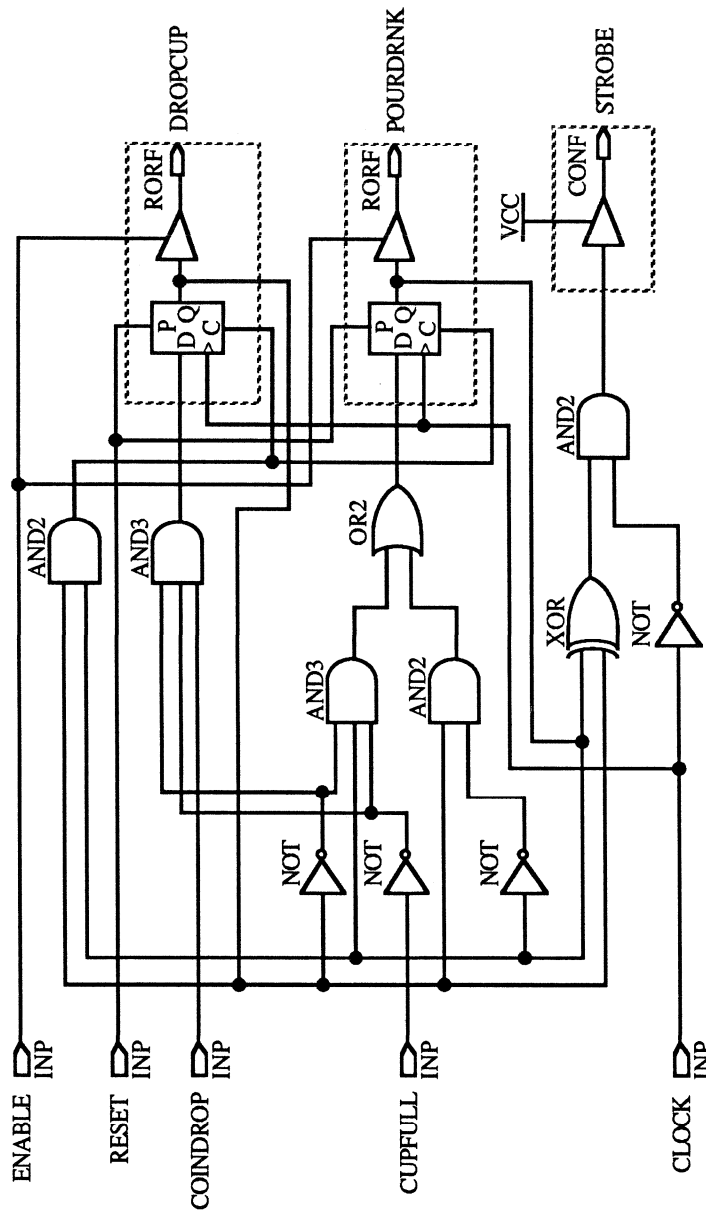


Figure BE-2. Sample Circuit

Entering the Sample Design

Step 1 — Choose the Text Editor:

Choose a standard ASCII text editor (in accordance with the specifications given in *General Requirements*).

The filename for this sample design is **BEVDIS.ADF**. (If you wish to establish a filename before entering your text editor, be sure to include the **.ADF** extension.)

Step 2 — Enter the Header:

Enter the Header Section. This section is optional, but is recommended because it provides design documentation that appears in the Logic Equation File, Utilization Report, and JEDEC File:

Your Name <Enter>
Your Company <Enter>
9/30/87 <Enter>
1.00 <Enter>
B <Enter>
EP310 <Enter>
Beverage Dispenser Controller <Enter>

Step 3 — Enter the Declarations:

Enter the Declarations Section, which specifies options, the target EPLD, and input and output pins:

<Enter> (Optional blank line to separate the Header Section from the Declarations Section.)

OPTIONS: SECURITY = OFF <Enter>
PART: EP310 <Enter>
**INPUTS: ENABLE@7, RESET@5, COINDROP,CUPFULL,
CLOCK <Enter>**
OUTPUTS: DROPCUP@14, STROBE, POURDRNK <Enter>

(Note that the inputs **ENABLE** and **RESET** and the output **DROPCUP** have declared pin assignments.)

Step 4 — Enter the Network:

Enter the Network Section. This section defines the input and I/O architecture to be programmed with primitive statements:

<Enter> (Optional blank line to separate the Network Section from the Declarations Section.)

NETWORK: <Enter> (<Enter> is optional)

```
ENABLE = INP(ENABLE) <Enter>  
RESET = INP(RESET) <Enter>  
COINDROP = INP(COINDROP) <Enter>  
CUPFULL = INP(CUPFULL) <Enter>  
CLOCK = INP(CLOCK) <Enter>  
DROPCUP,DROPCUP = RORF(DROPCUPd,CLOCK,  
    NEWCYCLE, RESET,ENABLE) <Enter>  
POURDRNK,POURDRNK = RORF(POURDRKd,CLOCK,  
    NEWCYCLE,RESET,ENABLE) <Enter>  
STROBE = CONF(STROBEc,) <Enter>
```

Note that no **Oe** term is given for **STROBE**: it will go to **VCC**—the default value. Information on primitive statement syntax is given in *Network Section Requirements*.

Step 5 — Enter the Equations:

Enter the Equations Section, which implements the Boolean logic for the design:

<Enter> (Optional blank line to separate the Network Section from the Equations Section.)

EQUATIONS: <Enter> (<Enter> is optional)

```
DROPCUPd = COINDROP * /DROPCUP
           * /POURDRNK; <Enter>
CUPREADY = DROPCUP * /POURDRNK; <Enter>
POURDRKd = CUPREADY
           + /CUPFULL * /DROPCUP
           * POURDRNK; <Enter>
STROBEc = /CLOCK * (CUPREADY
                + (/DROPCUP * POURDRNK)); <Enter>
NEWCYCLE = DROPCUP * POURDRNK; <Enter>
```

Step 6 — Terminate the ADF:

Terminate the ADF with the End Statement:

<Enter> (Optional blank line to separate the Equations Section from the End statement.)

END\$ <Enter>

Your ADF should look like the one shown in Figure BE-3. (Additional ADF samples are given in Figures BE-4 through BE-7.)

Step 7 — Save the Design:

Save your design under the legal DOS filename **BEVDIS.ADF** and return to DOS.

Your Name
Your Company
9/30/87
1.00
B
EP310
Beverage Dispenser Controller

OPTIONS: SECURITY = OFF
PART: EP310
INPUTS: ENABLE@7, RESET@5, COINDROP, CUPFULL, CLOCK
OUTPUTS: DROPCUP@14, STROBE, POURDRNK

NETWORK:

ENABLE = INP(ENABLE)
RESET = INP(RESET)
COINDROP = INP(COINDROP)
CUPFULL = INP(CUPFULL)
CLOCK = INP(CLOCK)
DROPCUP,DROPCUP = RORF(DROPCUPd,CLOCK,NEWCYCLE,
RESET,ENABLE)
POURDRNK,POURDRNK = RORF(POURDRKd,CLOCK,NEWCYCLE,
RESET,ENABLE)
STROBE = CONF(STROBEc,)

EQUATIONS:

DROPCUPd = COINDROP * /DROPCUP * /POURDRNK;
CUPREADY = DROPCUP * /POURDRNK;
POURDRKd = CUPREADY
+ /CUPFULL * /DROPCUP * POURDRNK;
STROBEc = /CLOCK * (CUPREADY
+ (/DROPCUP * POURDRNK));
NEWCYCLE = DROPCUP * POURDRNK;

ENDS

Figure BE-3. ADF for the Sample Design (BEVDIS)

Step 8 — Check the ADF:

To check the ADF for errors such as unconnected pins or faulty syntax, submit the completed ADF to the ADP. The ADP translates the ADF into internal logic equations, detects design and syntax errors, and displays appropriate error messages. (See *Step 9*.)

Step 9 — Process the Design:

Now you submit the **BEVDIS.ADF** file to the ADP. From DOS enter:

APLUS <Enter>

The APLUS Menu is displayed.

Press <F4> to display the ADP Menu.

The prompt asks you to specify your form of input. Press <Enter>. (ADF is the default.)

Now answer the <F4> **File Name(s)** prompt by typing:

BEVDIS <Enter>



You need not enter the filename extension; A+PLUS will automatically add the extension for you. Be sure to specify the correct pathname and directory.

You are then prompted through the remaining ADP Menu functions:

For <F5> **Minimization**, press <Enter>. (Y [Yes] is the default).

For <F6> **Inversion Control**, press <Enter>. (N [No] is the default).

For <F7> **LEF Analysis**, press <Enter>. (N [No] is the default).

After you have answered the sequential prompts of the ADP Menu, you are asked:

Do you wish to run under the above conditions [Y/N]?

Enter **Y** or press **<F8>** (**Execute**) to execute the ADP. During design processing, the ADP and its modules will display information messages that report current processing status. When the design cycle is completed, you are asked:

Would you like to implement another design [Y/N]?

Enter **N**. You are returned to the APLUS Menu. For detailed information on the Altera Design Processor and the ADP Menu functions, refer to *A+PLUS and ADP Reference* in the ***A+PLUS Reference Guide***.

Step 10 — Program the EPLD:

Finally, you submit your design to LogicMap II. While still in the APLUS Menu, press **<F5>** to select LogicMap II. If the Logic Programmer card is plugged in, the program will come up on the screen. If not, the following message is displayed:

Programmer self test failed

Device must NOT be in socket for this test to pass!

Enter:

C to continue without programming card

T to run diagnostics again

Q to return to operating system

When the LogicMap II System Level Window is displayed, you are asked to wait until the calibration process has been completed. Then the System Level HELP Window is opened.



Do not put the EPLD into the socket of the programming unit until you are prompted to do so.

Select **Program Device** with the box cursor and enter the filename **BEVDIS**. When you are prompted to:

Select Device for Programming

enter

EP310 <Enter>

LogicMap II automatically checks whether the EPLD is erased and ready for programming. After you have answered all the prompts, programming time is approximately five to ten seconds for this device. (For complete information on device programming, refer to the ***LogicMap II*** manual.)

This concludes the sample session.

Altera Design File (ADF) Format

After you have created a design with Boolean equations, you enter it into the ADF format. An ADF contains the following sections:

- Header Section
- Declarations Section
- Network Section
- Equations Section
- End Statement

The syntax requirements for each ADF section are described below. (For a complete Backus-Naur Form (BNF) description of the ADF, refer to *Altera Design File Format* in the ***A+PLUS Reference Guide***.)

Figure BE-4 shows a sample ADF that illustrates the ADF sections and the use of design conventions.

<i>Header Section</i>	DESIGNER NAME COMPANY NAME SEPT. 30, 1987 1.00 A EP310 7493 DIV8 COUNTER	White space is permitted between any lines
<i>Declarations Section</i>	OPTIONS: TURBO = OFF PART: EP320 INPUTS: RESET1, RESET2, CLOCK@1 OUTPUTS: Q0, Q1, Q2	Turbo-Bit set to OFF; Security Bit defaults to OFF Pin assignment
<i>Network Section</i>	NETWORK: CLOCK = INP(CLOCK) %CLOCK INPUT% RESET1 = INP(RESET1) RESET2 = INP(RESET2) Q2,Q2 = RORF(Q2d, CLOCK, CLEAR, GND, VCC) Q1,Q1 = RORF(Q1d, CLOCK, CLEAR, GND, VCC) Q0,Q0 = RORF(Q0d, CLOCK, CLEAR, GND, VCC)	User comment
<i>Equations Section</i>	EQUATIONS: Q0d = /Q0 ; Q1d = /Q0 * /Q1 + Q0 * Q1 ; Q2d = Q2*Q1 + Q0*Q2 + Q0'*Q1'*Q2' ; CLEAR = RESET1*RESET2 ;	Equations end with semicolon Equations may span lines
<i>End Statement</i>	END\$	All ADFs must terminate with END\$ statement

Figure BE-4. Sample ADF

Header Section Requirements

The Header Section provides the design documentation. Header Section requirements are as follows:

- The Header Section in the ADF is optional. However, since the file header serves to identify the Utilization Report and the JEDEC file, this section is highly recommended.
- If it is present, it must be the first section in the ADF.
- All fields in the Header Section are terminated by <Enter>.
- Any printable character except the asterisk (*) is allowed.
- Fields in the Header Section must appear in the following order, with one item on each line. The maximum character count for each field is indicated in parentheses.

Designer (48)

Company (60)

Date (24)

Number (24)

Revision (24)

EPLD (10)

Comment (512)

Any other information (may be more than one line)

Declarations Section Requirements

The Declarations Section specifies the EPLD used for the design, input and output pin names, and optional pin assignments. It also allows you to disable/enable the Turbo-Bit (a control bit for choosing speed and power characteristics of an EPLD) and the Security Bit (which prevents a device from being interrogated or inadvertently reprogrammed). This section contains subsections which are referred to as the Options, Part, Inputs, and Outputs sections. Declarations Section requirements are as follows:

Options Section

The Options Section is optional. It is identified with the keyword **OPTIONS:**, followed by either or both of the following option specifiers:

- **TURBO**, indicating Turbo-Bit. The values are **ON** (the default) or **OFF**. If **ON** or **OFF** is not specified, the Turbo-Bit defaults to **ON**. (The **BUSTER** (EPB1400) and **EP310** parts do not support the Turbo-Bit option. LogicMap will ignore any Turbo-Bit information entered for these EPLDs.)
- **SECURITY**, indicating the Security Bit. The values are **ON** or **OFF** (the default). If **ON** or **OFF** is not specified, the Security Bit defaults to **OFF**.

The keyword **OPTIONS:** must be the first word in the section. The string is terminated with **<Enter>**. Example:

```
OPTIONS: TURBO = ON, SECURITY = OFF <Enter>
```

Part Section

The Part Section is required. It consists of the keyword **PART:**, followed by the name of an Altera EPLD or **AUTO** (for automatic part selection). The keyword **PART:** must be the first word in the section. The string is terminated with **<Enter>**. Example:

```
PART: EP1210 <Enter>
```



Automatic part selection will not be successful if you specify pin assignments or if the design contains too many inputs, outputs, or macrocells.

Inputs Section

The Inputs Section is required. It consists of the keyword **INPUTS:**, followed by a list of all input pin names used in the design. The keyword **INPUTS:** must be the first word in the section. List elements are delimited by commas or white space. The list may span lines at any point except within a name and must be terminated with **<Enter>**. You may include specific pin assignments by appending an at-symbol (**@**) plus a one- or two-digit pin number to any input name on the list. (In the EP1800G, pin numbers are specified with an @-symbol plus one letter and a one- or two-digit number.) Pin numbers can be specified for any pin name. Pin names may contain up to eight characters, including any printable character except percent symbol (**%**), comma (**,**), equals sign (**=**), at-symbol (**@**), or left and right parentheses (**(**) and **)**). The @-symbol plus pin number may contain up to three characters. Example:

```
INPUTS: ENABLE@7, RESET@5, COINDROP, CUPFULL,  
CLOCK@1 <Enter>
```



1. If you use an asterisk (*) in an input or output pin name, it will be converted into a tilde (~) in the JEDEC file generated by the ADP. You must ensure that this conversion will not create duplicate pin names (for example, if you use both * and ~ when naming pins).
2. The ADP creates internal node names that contain periods (.). User-assigned pin names that contain periods may occasionally conflict with these node names and cause unpredictable results.
3. The Functional Simulator ignores all user-defined pin names that contain periods (periods are allowed only for referencing internal nodes of I/O primitives).

Outputs Section

The Outputs Section is required. It consists of the keyword **OUTPUTS:**, followed by a list of all output pin names used in the design. The keyword **OUTPUTS:** must be the first word in the section. Output pin names and numbers have the same format as in the Inputs Section. (Note: Pin numbers can be specified for any pin name or buried register output.) Example:

OUTPUTS: DROPCUP, POURDRNK@16, STROBE <Enter>



1. Bidirectional pins using the **ROIF**, **TOIF**, and **COIF** primitives must be declared only in the Outputs Section of the Altera Design File (ADF), not in the Inputs Section.
2. Buried register outputs (of **NOCF**, **NOJF**, **NORF**, and **NOTF** primitives) may be listed in the Outputs Section. They may also be assigned to the pins associated with EP1800 global macrocells and BUSTER generic macrocells. However, these node names must be different from true output pin names.

Network Section Requirements

The required Network Section specifies the input and I/O architecture to be programmed. Network Section requirements are as follows:

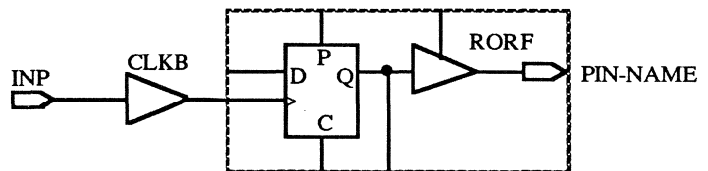
- The keyword **NETWORK:** must be the first word in the section.
- Following the keyword, the architecture of each input, output, and I/O pin used in the design must be defined with primitive statements. Primitive statements must be separated by white space or **<Enter>**. Table BE-1 shows the syntax for each element with mnemonic node names. (Refer also to *Altera Primitive Library* in the **A+PLUS Reference Guide** for the formal ADF syntax of primitive statements, and to *Naming Conventions* [below].)
- In addition to dedicated input pins, any I/O pin can be used as an input pin with the **INP** primitive.

- The Clear (C), Preset (P), and Output Enable (Oe) inputs to I/O primitives are optional and do not require connections. If they are left unconnected, you must retain the commas that delimit the fields. Example:

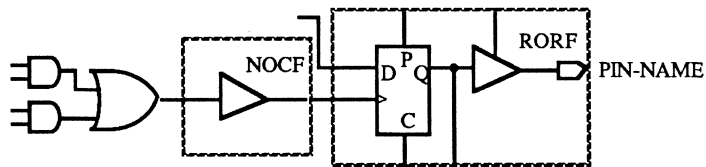
Namep, Name = RORF(Named, Clk,,,)

The reserved signal names VCC and GND are then substituted by default (C, P = GND; Oe = VCC), i.e., Clear and Preset signals are disabled and the output is permanently enabled.

- The Clock (Clk), Read Strobe (Rs) and Write Strobe (Ws) inputs to Bus I/O primitives (namely, BUSX, LBUSI, LBUSO, LIMP8, RBUSI and RIMP8) are optional. If they are connected, they must be connected directly to a pin or to VCC, GND, and GND, respectively. If they are left unconnected, you must retain the commas that delimit the fields. When the Clk, Rs, and Ws are left unconnected, logic must be connected to the Output Latch Enable (Ole), Output Enable (Oe), and Write Enable (We), respectively, which will have full control of the flipflop. (For additional information, refer to the Bus I/O primitive descriptions in *Altera Primitive Library* in the *A+PLUS Reference Guide* and to the *EPB1400 Data Sheet*.)
- The Clk inputs to primitives can be obtained in three ways (legal clocking configurations for all Altera EPLDs are described in *Appendix A* of the *A+PLUS Reference Guide*):
 - (1) A clock driven by an input pin is assumed to be synchronous unless it is specifically assigned to a pin other than a dedicated synchronous clock pin.
 - (2) To force a clock input to be asynchronous, feed the signal to a CLKB (Asynchronous Clock Buffer) primitive and connect the CLKB output to the input of a flipflop. Example:

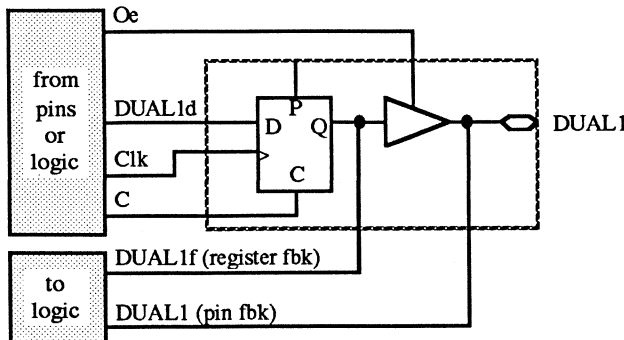


- (3) A clock driven by Boolean logic is always asynchronous. (Use of the `CLKB` primitive is optional.) BUSTER (EPB1400) macrocells allow Boolean equations feeding the Clock inputs to I/O primitives to contain up to two product terms. In the EP600, EP610, EP900, EP910, and EP1800 parts, the logic feeding an asynchronous clock may contain one product term. However, in any of these EPLDs, asynchronous clocks with logic that requires up to eight product terms as input can be implemented by connecting the logic as an input to an `NOCF` primitive, and using the output of the `NOCF` as the clock input to the register. Example:



- Dual I/O feedback is supported by EP1800 global macrocells and BUSTER generic macrocells. To implement dual I/O feedback, you enter the same pin name in both the Inputs and Outputs Section, and use this pin name in both input and I/O primitives in the Network Section. Pin assignments are optional; if they are used, both pin names must be assigned to the same pin.

The following example shows a partial ADF that implements dual I/O feedback for the pin called `DUAL1`:



INPUTS:

DUAL1 ... % pin name in Inputs Section%

OUTPUTS:

DUAL1 ... % pin name also in Outputs Section%

NETWORK:

DUAL1 = INP(DUAL1)

DUAL1, DUAL1f = RORF(DUAL1d, Clk, C, , Oe)

% DUAL1f is internal registered feedback (before tri-state buffer) DUAL1 output is pin feedback (after tri-state buffer) %

EQUATIONS:

Oe = A * B; % output enable function %

DUAL1d = C * D; % DUAL1d may also be a function of DUAL1%

- Buried register outputs (of NOCF, NOJF, NORF, and NOTF primitives) may be assigned to the pins associated with EP1800 global macrocells and BUSTER generic macrocells. When logic is buried on these macrocells, the associated pin may also be used as an input pin.
- The procedure for defining Active Low inputs and outputs is described in *Active Low Signals* (below).

Table BE-1. Input, I/O, and Bus I/O Syntax (Part 1 of 3)

Legend			
C	Clear	Ole	Output latch enable
Clk	Clock	P	Preset
Ile	Input latch enable	Re	Read enable
Namebp	dedicated bus port name	Rs	Read strobe
Nameibus	Internal bus name	We	Write enable
Namep	Pin name	Ws	Write strobe
Oe	Output enable		

Input Primitives:	
Name =	INP(Namep) [Input]
Name =	LINP(Namep,Ile) [Latched Input]
I/O Primitives:	
Namep,Name =	COCF(Namec,Oe) [Combinatorial Output,Combinatorial Feedback]
Namep,Name =	COIF(Namec,Oe) [Combinatorial Output,I/O Feedback]
Namep,Name =	COLF(Namec,Oe,Ile) [Combinatorial Output,Latched Feedback]
Namep =	CONF(Namec,Oe) [Combinatorial Output,No Feedback]
Namep,Name =	CORF(Namec,Clk,C,P) [Combinatorial Output,Registered Feedback]

Table BE-1. Input, I/O, and Bus I/O Syntax (Part 2 of 3)

I/O Primitives: <i>(continued)</i>	
Namep,Name =	JOJF(Namej,CIk,Namek,C,P,Oe) [JK Output,JK Feedback]
Namep =	JONF(Namej,CIk,Namek,C,P,Oe) [JK Output,No Feedback]
Name =	NOCF(Namec) [No Output,Combinatorial Feedback]
Name =	NOJF(Namej,CIk,Namek,C,P) [No Output,JK Feedback]
Name =	NORF(Named,CIk,C,P) [No Output,Registered Feedback]
Name =	NOSF(Names,CIk,Namer,C,P) [No Output,SR Feedback]
Name =	NOTF(Namet,CIk,C,P) [No Output,T Feedback]
Namep,Name =	ROCF(Named,CIk,C,P,Oe) [Registered Output,Combinatorial Feedback]
Namep,Name =	ROIF(Named,CIk,C,P,Oe) [Registered Output,I/O Feedback]
Namep,Name =	ROLF(Named,CIk,C,P,Oe,Ile) [Registered Output,Latched Feedback]
Namep =	RONF(Named,CIk,C,P,Oe) [Registered Output,No Feedback]
Namep,Name =	RORF(Named,CIk,C,P,Oe) [Registered Output,Registered Feedback]
Namep =	SONF(Names,CIk,Namer,C,P,Oe) [SR Output,No Feedback]
Namep,Name =	SOSF(Names,CIk,Namer,C,P,Oe) [SR Output,SR Feedback]

Table BE-1. Input, I/O, and Bus I/O Syntax (Part 3 of 3)

I/O Primitives: <i>(continued)</i>	
Namep,Name =	TOIF(Name _t ,Clk,C,P,Oe) [T Output,I/O Feedback]
Namep =	TONF(Name _t ,Clk,C,P,Oe) [T Output,No Feedback]
Namep,Name =	TOTF(Name _t ,Clk,C,P,Oe) [T Output,T Feedback]
Bus I/O Primitives:	
Name ₀ ,Name ₁ ,Name ₂ ,Name ₃ , Name ₄ ,Name ₅ ,Name ₆ ,Name ₇ =	LINP8(Name _{0p} ,Name _{1p} ,Name _{2p} ,Name _{3p} , Name _{4p} ,Name _{5p} ,Name _{6p} ,Name _{7p} ,Ws,We) [8-Bit Latched Input] †
Name ₀ ,Name ₁ ,Name ₂ ,Name ₃ , Name ₄ ,Name ₅ ,Name ₆ ,Name ₇ =	RINP8(Name _{0p} ,Name _{1p} ,Name _{2p} ,Name _{3p} , Name _{4p} ,Name _{5p} ,Name _{6p} ,Name _{7p} ,Ws,We) [8-Bit Registered Input] †
Name _{0bp} ,Name _{1bp} ,Name _{2bp} ,Name _{3bp} , Name _{4bp} ,Name _{5bp} ,Name _{6bp} ,Name _{7bp} =	BUSX(Name _{ibus} , Rs,Oe) [Bus Transceiver]
Name ₀ ,Name ₁ ,Name ₂ ,Name ₃ , Name ₄ ,Name ₅ ,Name ₆ ,Name ₇ =	LBUSI(Ibus,Ws,We) [Latched Bus Input to Logic]
Name _{ibus} =	LBUSO(Name _{0d} ,Name _{1d} ,Name _{2d} ,Name _{3d} , Name _{4d} ,Name _{5d} ,Name _{6d} ,Name _{7d} ,Clk,Ole,Re) [Latched Bus Output from Logic]
Name ₀ ,Name ₁ ,Name ₂ ,Name ₃ , Name ₄ ,Name ₅ ,Name ₆ ,Name ₇ =	RBUSI(Name _{ibus} ,Ws,We)

Notes:

† Each pin name corresponds to the node name of the same number, e.g., Name₀ corresponds to Name_{0p}.

Naming Conventions

This section describes naming conventions we recommend for defining the Network Section of a design. Tables BE-1 and BE-2 illustrate the recommended naming conventions. (Special naming conventions for Active Low signals are discussed in the next section.) Table BE-3 shows legal pin and node name characters.

Inputs

Name_p represents a device pin name. It must have a corresponding entry in the Inputs or Outputs Section. **Name** represents an input to the device logic array. It appears on the right-hand side of equations. Whenever possible, the pin name should be the same as the array input name (i.e., node name) to allow pin names to be used directly in Boolean equations, e.g., $x = \text{INP}(x)$.

I/O (No Feedback)

The input to the I/O primitive should be the same as the output pin name with **c**, **d**, **j**, **k**, **r**, **s**, or **t** appended to represent the output type. (The last letter indicates combinatorial feedback or the flipflop type.) Each input must appear exactly once as the left-hand side of an equation.

I/O (With Feedback)

The pin name should be the same as the I/O feedback node name, allowing the pin name to be used directly in the Boolean equations. The input to the I/O primitive should be the same as the output pin name with **c**, **d**, **j**, **k**, **r**, **s**, or **t** appended to represent the output type. (The last letter indicates combinatorial feedback or the flipflop type.) Each input must appear exactly once as the left-hand side of an equation. If dual I/O feedback is used, the feedback node name should be the same as the input and output pin names with **f** appended to represent internal registered feedback.

I/O (No Output)

The input to the I/O primitive should be the same as the primitive feedback node name with **c**, **d**, **j**, **k**, **r**, **s**, or **t** appended to represent the feedback to the logic array. (The last letter indicates combinatorial feedback or the flipflop type.) Each input must appear exactly once as the left-hand side of an equation. Note: if buried register output names are assigned to a pin, they may not conflict with other output pin names.

Bus I/O Primitives

Name0p through **Name7p** and **Name0bp** through **Name7bp** represent device pin names. You are not required to use the numbers 0 through 7; they merely indicate the relative position of the signal on the bus. In the **LINP8** and **RINP8** primitives, **Name0** through **Name7** represent the node names that correspond to the pins **Name0p** through **Name7p**, respectively. **Nameibus** represents the name for the internal 8-bit wide bus. Each input must appear exactly once as the left-hand side of an equation.

Table BE-2. Recommended Naming Conventions

TYPE	SYNTAX
Input	Name = INP(Namep) <i>Example:</i> ENABLE = INP(ENABLE)
I/O (No Feedback)	Name = RONF(Named,Clk,C,P,Oe) <i>Example:</i> CUPFULL = RONF(CUPFULLd,CLOCK, GND,GND,VCC)
I/O (With Feedback)	Namep,Name = RORF(Named,Clk,C,P,Oe) <i>Example:</i> DROPCUP,DROPCUP = RORF(DROPCUPd, CLOCK,GND,GND,VCC)
I/O (No Output)	Name = NOCF(Namec) <i>Example:</i> ABBY = NOCF(ABBYc)
<p>Note: The maximum length for any name is eight characters. Both capitals and lowercase are allowed and significant, i.e., "NODEA" is not equal to "nodeA." Names may not span lines.</p>	

Table BE-3. ADF Legal Pin Name and Node Name Characters

ADF Legal Pin Name Characters				ADF Legal Node Name Characters		
!	:	S	j	A	X	u
"	;	T	k	B	Y	w
#	<	U	l	C	Z	x
\$	>	V	m	D	a	y
&	?	W	n	E	b	z
'	A	X	o	F	c	0
*	B	Y	p	G	d	1
+	C	Z	q	H	e	2
-	D	[r	I	f	3
.	E	\	s	J	g	4
/	F]	t	K	h	5
0	G	^	u	L	i	6
1	H	_	v	M	j	7
2	I	`	w	N	k	8
3	J	a	x	O	l	9
4	K	b	y	P	m	
5	L	c	z	Q	n	
6	M	d	{	R	o	
7	N	e		S	p	
8	O	f	}	T	q	
9	P	g	~	U	r	
	Q	h		V	s	
	R	i		W	t	

Active Low Signals

To specify Active Low inputs and outputs, we recommend that you adhere to the following convention (an example is given below):



The Read Strobe and Write Strobe input pins (/CRS and /CWS) for BUSTER (EPB1400), as well as the Rs and Ws inputs to Bus I/O primitives, are predefined as Active Low signals. Therefore, it is not necessary to perform the signal inversion described below to implement Active Low logic.

- (1) In the Inputs or Outputs Section, identify the Active Low signal by prefixing an 'n' to the pin name (e.g., "nSIGX").
- (2) Specify an intermediate node name that retains the 'n' in the pin name (e.g., nSIGX = INP(nSIGX)).
- (3) Invert the intermediate node and drop the 'n' to obtain "SIGX", (e.g., SIGX = NOT(nSIGX)).
- (4) Use "SIGX" in the Equations Section.

By following this procedure, you will implement an Active Low input or output. Example:

INPUTS:

nSIGIN % Active Low input %

OUTPUTS:

nSIGOUT % Active Low output %

NETWORK:

nSIGIN = INP(nSIGIN) % Creates an intermediate node %

SIGIN = NOT(nSIGIN) % Inverts the input %

nSIGOUT = CONF(nSIGOUTc,) %Creates an inverted intermed. node%

nSIGOUTc = NOT(SIGOUT) % Passes the inverted node to output %

EQUATIONS:

A = SIGIN * B; % When the pin is low, SIGIN is true %

SIGOUT = C + D % When SIGOUT is true, the pin is low%



It is also possible to identify an Active Low input or output by attaching a slash (/), exclamation point (!), or single quote (') to the pin name (e.g., /SIGX, !SIGX, SIGX'). Note, however, that these characters function as NOT-operators only when used in Boolean equations. Since node names must be alphanumeric (and therefore may not include the /, !, or '), the intermediate node name defined in the Network Section should be of the form "nSIGX" (e.g., nSIGX = INP(/SIGX)).

Equations Section Requirements

The Equations Section implements Boolean logic. Equations Section requirements are as follows:

- The keyword **EQUATIONS:** must be the first word in the section.
- Each equation must have exactly one node name on the left side (possibly complemented), followed by an equals sign (=), a Boolean expression, a semicolon (;), and <Enter>.
- Equations may span lines at any point except within a name.
- The fan-out of each Boolean equation is unlimited.
- Boolean equations need not be in sum-of-products form. Parentheses are used to indicate grouping.
- The equations may use any of the following operators:

/	or !	or '	for NOT
*	or &		for AND
+	or #		for OR
()			for establishing precedence

The / and ! NOT-operators are prefixed and the ' NOT-operator is postfixed. Example:

SIGOUT = IN1 * /IN2 * !IN3 * (IN4' + IN5);

- Entire equations may be substituted into the right-hand sides of other equations by substituting intermediate variables.

The following example uses **TERM** as an intermediate variable (note that **TERM** need not be defined before being used):

$$\begin{aligned} D1 &= Oa1 * Oa2 * (Fbk1' + Fbk2) + TERM; \\ TERM &= Fbk2 + Oa1 * Fbk1; \end{aligned}$$

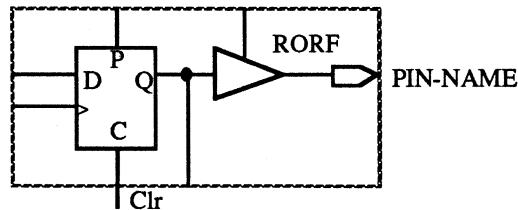
The compiler converts this to:

$$\begin{aligned} D1 &= Oa1 * Oa2 * (Fbk1' + Fbk2) + Fbk2 \\ &+ Oa1 * Fbk1 \end{aligned}$$

- BUSTER** (EPB1400) macrocells allow Boolean equations feeding the Clock (**Clk**) and Output Enable (**Oe**) inputs to I/O primitives to contain up to two product terms; Clear (**C**) inputs are limited to one product term. The logic feeding the Output Latch Enable (**Ole**), Output Enable (**Oe**), Read Enable (**Re**), and Write Enable (**We**) inputs to Bus I/O primitives (namely, **BUSX**, **LBUSI**, **LBUSO**, **LINP8**, **RBUSI** and **RINP8**) may contain up to two product terms after minimization has taken place.

The architecture of other Altera parts allows Boolean equations feeding the Clear, Clock, Preset, Latch Enable, and Output Enable inputs to I/O primitives (i.e., **C**, **Clk**, **P**, **Le**, and **Oe**) to contain only one product term. See *Appendix A* in the **A+PLUS Reference Guide** for specific information.

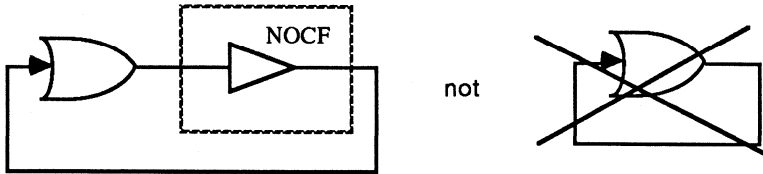
Example:



$$Clr = Oa1 * Fbk2' * Oa3;$$

- Boolean equations cannot feed back to themselves without first going through an I/O primitive or buried register. This means that the left-hand side of an equation can never appear in the right-hand side or in an intermediate term of the right-hand side of the

same equation. For example, cross-coupled latches cannot be constructed within a macrocell using only Boolean expressions. To obtain combinatorial feedback, you must use an I/O primitive (e.g., COCF; NOCF, ROCF, COIF, ROIF, TOIF). Example:



End Statement

This statement terminates the ADF. It consists of the following string:

END\$ <Enter>

Additional Guidelines

In addition to the conventions and requirements for the individual ADF sections, you should note the following guidelines:

- White space or comments may be inserted between syntax elements to make reading easy. Comments are enclosed by percent symbols (%).
- `<CR><LF>` is equivalent to `<Enter>`.
- Assigning pin numbers to both inputs and outputs may speed the fitting process.
- Automatic part selection will not be successful if you specify pin assignments or if the design contains too many inputs, outputs, or macrocells.
- Combinatorial feedback from a macrocell to itself may cause unpredictable results.
- The Altera Design Processor does not support the product-term sharing feature of the EP1210. This feature is accessed through LogicMap II.
- A+PLUS offers two special features: (1) Turbo-Bit and (2) Security Bit.
 - (1) The Turbo-Bit is a control bit for choosing speed and power characteristics of an EPLD. It can be set to **ON** or **OFF** in the Declarations Section. If the Turbo-Bit status is not specified, it will default to **ON**. Before programming an EPLD, LogicMap also allows you to toggle the Turbo-Bit **ON** or **OFF**. (Note: the BUSTER and EP310 parts do not support the Turbo-Bit option. LogicMap will ignore Turbo-Bit information entered for these EPLDs.)
 - (2) The Security Bit prevents a device from being interrogated or inadvertently reprogrammed. It can be set to **ON** or **OFF** in the Declarations Section. If the status of the Security Bit is not specified, it will default to **OFF**. Before programming an EPLD, LogicMap prompts you to indicate whether you wish to turn the Security Bit feature **ON** or **OFF**.

Figures BE-5, BE-6, and BE-7 show sample ADFs for the EP1210, EP600, and EP320, respectively.


```

7445 1-OF-10 DECODER
OPTIONS: TURBO = OFF
PART: EP1210
INPUTS:
A3, A2, A1, A0
OUTPUTS:
/O9, /O8, /O7, /O6, /O5, /O4, /O3, /O2, /O1, /O0
NETWORK:
A3 = INP(A3) % Input Signal %
A2 = INP(A2) % Input Signal %
A1 = INP(A1) % Input Signal %
A0 = INP(A0) % Input Signal %
/O9 = CONF(nO9c,) nO9c = NOT(O9)
/O8 = CONF(nO8c,) nO8c = NOT(O8)
/O7 = CONF(nO7c,) nO7c = NOT(O7)
/O6 = CONF(nO6c,) nO6c = NOT(O6)
/O5 = CONF(nO5c,) nO5c = NOT(O5)
/O4 = CONF(nO4c,) nO4c = NOT(O4)
/O3 = CONF(nO3c,) nO3c = NOT(O3)
/O2 = CONF(nO2c,) nO2c = NOT(O2)
/O1 = CONF(nO1c,) nO1c = NOT(O1)
/O0 = CONF(nO0c,) nO0c = NOT(O0)

EQUATIONS:
O0 = A3' & A2' & A1' & A0' ;
O1 = A0 & A3' & A2' & A1' ;
O2 = A1 & A0' & A2' & A3' ;
O3 = A0 & A1 & A2' & A3' ;
O4 = A2 * A0' * A1' * A3' ;
O5 = A2 * A0 * A1' * A3' ;
O6 = A2 * A1 * A0' * A3' ;
O7 = A2 * A1 * A0 * A3' ;
O8 = A3 * A2' * A1' * A0' ;
O9 = A3 * A0 * A2' * A1' ;

END$

```

Header Section is optional
 Turbo-Bit set to OFF; Security Bit defaults to OFF
 Physical pin assignments not specified
 Output Enable set to default; inverted node names give Active Low outputs
 Different Boolean operators allowed

Figure BE-5. Sample ADF for EP1210

DESIGNER NAME
 COMPANY NAME
 9/30/87
 1
 A
 EP600
 4 BIT COUNTER

PART: EP600
 INPUTS: CLOCK@2, !HOLD@11
 OUTPUTS: Q1, Q2, Q3, Q4

NETWORK:

CLOCK = INP(CLOCK)
 CLOCKa = CLKB(ASY)
 nHOLD = INP(!HOLD) % Intermed node %
 HOLD = NOT(nHOLD) % Active Low %
 Q1,Q1 = TOTF(D1t,CLOCKa,,)
 Q2,Q2 = TOTF(D2t,CLOCKa,,)
 Q3,Q3 = TOTF(D3t,CLOCKa,,)
 Q4,Q4 = TOTF(D4t,CLOCKa,,)

% HOLD = LOW THEN STOP COUNT
 HOLD = HIGH THEN COUNT %

EQUATIONS:

D1t = /HOLD;
 D2t = Q1;
 D3t = Q1 * Q2;
 D4t = Q1 * Q2 * Q3;

ASY = /HOLD * CLOCK;

END\$

Active Low input symbol

Pin assignment

*Turbo-Bit and Security Bit status
 not declared (default to ON and
 OFF, respectively)*

Active High outputs

*Asynchronous clock (use of
 CLKB primitive is optional)*

*User comments delimited
 by % symbols*

Gated clock (1 product term)

Figure BE-6. Sample ADF for EP600

SEPTEMBER 30, 1987

P7022

A

EP320 4-BIT COUNTER WITH 2 INPUT MUX EPLD DESIGN SPECIFICATION

OPTIONS: TURBO = OFF, SECURITY = ON

PART: EP320

INPUTS: CLK@1, A0@2, A1@3, A2@4, A3@5, B0@6, B1@7,
B2@8, B3@9, nOE@11, I0@18, I1@13, C1@19

All inputs and
outputs have
assigned
pin numbers

OUTPUTS: CO@12, Q3@14, Q2@15, Q1@16, Q0@17

NETWORK:

CLK = INP(CLK) I1 = INP(I1) C1 = INP(C1)
I0 = INP(I0) A1 = INP(A1) A2 = INP(A2) A3 = INP(A3)
A0 = INP(A0) B1 = INP(B1) B2 = INP(B2) B3 = INP(B3)
B0 = INP(B0) nOE = INP(nOE) OE = NOT(nOE)
Q0,Q0 = RORF(Q0d,CLK,,OE)
Q1,Q1 = RORF(Q1d,CLK,,OE)
Q2,Q2 = RORF(Q2d,CLK,,OE)
Q3,Q3 = RORF(Q3d,CLK,,OE)
CO = CONF(COc,VCC)

Multiple Network
Statements on
the same line

Output Enable
control (1
product term)

Clear and Preset
set to default
condition

EQUATIONS:

$$/Q0d = /I1*/I0*Q0+/I1*I0*/A0+I1*/I0*/B0+I1*I0*/C1*/Q0 \\ + I1*I0*C1*Q0 ;$$

Inverting left-
hand side of
equation is
equivalent to
inverting all of
right-hand side

$$/Q1d = /I1*/I0*/Q1+/I1*I0*/A1+I1*/I0*/B1+I1*I0*/C1*/Q1 \\ + I1*I0*/Q0*/Q1+I1*I0*C1*Q0*Q1 ;$$

Figure BE-7. Sample ADF for EP320 (Part 1 of 2)

```

/Q2d = /I1*/I0*/Q2          % HOLD Q2 %
      + /I1*I0*/A2          % LOAD A2 %
      + I1*/I0*/B2          % LOAD B2 %
      + I1*I0*/C1*/Q2       % HOLD Q2 IF NO CARRY IN %
      + I1*I0*/Q0*/Q2       % HOLD Q2 IF Q0 = L %
      + I1*I0*/Q1*/Q2       % HOLD Q2 IF Q1 = L %
      + I1*I0*C1*Q0*Q1*Q2 ; % COUNT IF CARRY IN AND
                               Q0,Q1,Q2 = H %
/Q3d = /I1*/I0*/Q3          % HOLD Q3 (MSB) % ←
      + /I1*I0*/A3          % LOAD A3 %
      + I1*/I0*/B3          % LOAD B3 %
      + I1*I0*/C1*/Q3       % HOLD Q3 IF NO CARRY IN %
      + I1*I0*/Q1*/Q3       % HOLD Q3 IF Q0 = L %
      + I1*I0*/Q2*/Q3       % HOLD Q3 IF Q1 = L %
      + I1*I0*C1*Q0*Q1*Q2*Q3 ; % HOLD Q3 IF Q2 = L %
                               % COUNT IF CARRY IN AND
                               Q0,Q1,Q2,Q3 = H %

/COc = /CI + /Q0 + /Q1      % CARRY OUT IF CARRY IN %
      + /Q2 + /Q3 ;         % AND Q0,Q1,Q2,Q3 = H %

ENDS$

```

*Equations
may span
several
lines*

Figure BE-7. Sample ADF for EP320 (Part 2 of 2)

Functional Simulator
Version 2.5
September 1987

P25-01769-02

Changes are made periodically to the information contained in this manual. These changes will be incorporated into subsequent editions.

Altera Corporation
3525 Monroe Street
Santa Clara, CA 95051
(408) 984-2800
TELEX: 888496

Copyright © 1985, 1986, 1987 Altera Corporation. All rights reserved.

No part of this manual may be copied or reproduced in any form or by any means without the prior written permission of Altera Corporation.

A+PLUS, SAM+PLUS, LogicMap, Turbo-Bit, MacroMuncher, SAM, BUSTER, EP310, EP320, EP600, EP610, EP900, EP910, EP1210, EP1800, EPB1400, EPS444, and EPS448 are trademarks of Altera Corporation. LogiCaps is a registered trademark of Altera. WordStar is a registered trademark of MicroPro Corporation. Fido is a trademark of Tom Jennings. MS-DOS is a trademark of Microsoft Corporation. FutureNet DASH is a trademark of FutureNet Corporation. PC-CAPS and PC-LOGS are trademarks of Personal CAD Systems, Inc. IBM Personal Computer is a registered trademark of International Business Machines Corporation.

Read This First...

The documentation for the Functional Simulator version 2.5 contains the following sections:

- Functional Simulator (FSIM)
- FSIM Reference
- Virtual Logic Analyzer
- FSIM Messages
- FSIM Glossary

Please insert them in your ***A+PLUS User Guide*** after the tab **III. Simulation.**

At the back of the package, you will find a Warranty Card. Please fill out the Warranty Card, insert it into the Registration Envelope with the Altera Registration Card, and mail it to Altera. You will receive future update information for Functional Simulator software *only* if you mail this card.



Version 2.5 of the Functional Simulator supports JEDEC files generated by A+PLUS 5.0. JEDEC files created by earlier versions of A+PLUS are not compatible with FSIM version 2.5.

Manual Updates

Altera documentation is updated with Change Pages, Section Reprints, and a **READ.ME** file.

Change Pages are issued for minor changes to the manual. New information is identified with vertical change bars in the margins next to the changed text. In addition, the date of issue is printed at the bottom of each page.

Section Reprints are issued if a section requires a substantial number of changes. The date of issue is indicated at the bottom of each page.

A **READ.ME File** is provided on the **A+PLUS INSTALL** diskette. This file contains information about recent changes to the Functional Simulator software that are not yet reflected in the manual.

Contents

Read This First	iii
Manual Updates	v

Functional Simulator

Functional Description	FS-3
The Simulation Process	FS-6
Phase 1	FS-8
Phase 2	FS-9
Phase 3	FS-10
Simulator Inputs.....	FS-11
JEDEC File Input	FS-11
Vector File Input	FS-11
Legal Logic Level Characters.....	FS-12
Legal Input Vector Format	FS-12
Pattern Format.....	FS-12
Table Format	FS-13
Command File Input	FS-14
Simulator Outputs.....	FS-15
Log File.....	FS-15

Functional Simulator (Continued)

Watch File	FS-15
Plot File.....	FS-15
Save File.....	FS-16
Sample Session — Interactive Mode.....	FS-17
Phase 1 — Setup Commands.....	FS-19
Phase 2 — Control Commands.....	FS-27
Phase 3 — Execution Commands.....	FS-29
Sample Session — Batch Mode.....	FS-37

FSIM Reference

FSIM Reference	FS-41
Vector Processing and the CYCLE Command	FS-42
Predefined Input Vector Sequences	FS-46
Binary Counting Sequence.....	FS-46
Rotating Bit Sequence.....	FS-46
Gray Code Sequence	FS-47
Glitch Generator Sequence.....	FS-47
Propagation of Undefined Node Levels	FS-48
Logic Functions.....	FS-48
Flipflops and Latches.....	FS-48
Output Enables	FS-48
Bidirectional Pins	FS-49
Bus Port Pins	FS-50
Externally Connected Clock Pins.....	FS-51
Referencing Predefined Node Names	FS-51
Referencing Subnodes	FS-52
Reserved Words.....	FS-53
Command Reference.....	FS-54
Command Format	FS-55
Node List	FS-55
Node Value List	FS-55
Cycle Value.....	FS-56
Command List	FS-56
Pathname	FS-56
BEGIN.....	FS-58
BREAK.....	FS-59
CLEAR	FS-61
CONTINUE (CONT).....	FS-62
CYCLE.....	FS-63
DESCRIBE (DESC).....	FS-64
DISPLAY (DISP)	FS-65
DOS	FS-66

ECHO	FS-67
EXECUTE (EXEC)	FS-68
FORCE	FS-69
GROUP	FS-70
HELP	FS-71
INITIALIZE (INIT)	FS-72
LOGFILE (LOG)	FS-73
PATTERN (PAT)	FS-74
PLOT	FS-75
QUIT	FS-77
RESTORE (REST)	FS-78
SAVE	FS-79
SIMULATE (SIM)	FS-80
STATUS	FS-81
SYMBOLS (SYMB)	FS-82
VECTOR (VEC)	FS-83
VIEW	FS-84
WATCH	FS-85

Virtual Logic Analyzer

Functional Description	LA-2
Invoking the Virtual Logic Analyzer	LA-4
Full Window	LA-5
Window Display Control	LA-7
Zooming	LA-7
Panning	LA-8
Node Cursor	LA-8
Waveform Cursors	LA-9
Searching	LA-11
Splitting the Window	LA-11
Node and Bus Control	LA-14
Nodes	LA-14
Buses	LA-15
Summary of VIEW Subcommands	LA-17
Window Manipulation Commands	LA-19
Waveform Manipulation Commands	LA-21

FSIM Messages

Error Messages	Messages-2
Warning Messages	Messages-12

FSIM Glossary

Illustrations

Figure		Page
FS-1.	The Functional Simulator.....	FS-5
FS-2.	The Simulation Process	FS-7
FS-3.	BEVDIS Sample Circuit	FS-18
FS-4.	Manually Created Design Input and Output Waveforms	FS-20
FS-5.	Pattern Format Vector File for BEVDIS	FS-21
FS-6.	Table Format Vector File for BEVDIS.....	FS-22
FS-7.	Input to the Keyboard	FS-31
FS-8.	Screen Output	FS-32
FS-9.	Output to the Log File BEVDIS.LOG.....	FS-33
FS-10.	Output to the Plot File BEVDIS.WAV	FS-34
FS-11.	Output to Watch File WATCH.OUT.....	FS-35
FS-12.	Command File BEVDIS.CMD for Batch Mode.....	FS-38
LA-1.	Virtual Logic Analyzer.....	LA-2
LA-2.	Full Window Screen.....	LA-5
LA-3.	Node and Waveform Cursors	LA-8
LA-4.	Relative Cursor Location	LA-10
LA-5.	Split Window	LA-12
LA-6.	Bus Waveform.....	LA-16

Tables

Table		Page
FS-1.	Simulation Commands	FS-57
LA-1.	VIEW Subcommands.....	LA-17

Functional Simulator

This package describes the Altera Functional Simulator and the process of simulating a design created with A+PLUS software. You will find:

- A functional description
- A description of the simulation process
- A detailed description of required inputs and outputs
- A sample session that guides you through the simulation process (interactive and batch modes)
- A reference section describing all available commands and advanced features
- A section describing the features and use of the Virtual Logic Analyzer
- A glossary of simulation terms

- A list of Functional Simulator error and warning messages, including suggestions for corrective action



Refer to *Installation* in the **A+PLUS User Guide** for instructions on how to install the Functional Simulator.

Functional Description

Altera's Functional Simulator software is a tool for testing the logical operation of your EPLD design. It uses specified design and part information to model the operation of an Altera EPLD before the design is actually committed to hardware.

The Functional Simulator uses the following input files:

- (1) A JEDEC File (with the extension **.JED**) as generated by the Altera Design Processor.
- (2) A file containing vectors (with the extension **.VEC**) that are used to simulate input logic levels for the EPLD. The Vector File is a "1's and 0's file" describing the input conditions.
- (3) A file containing commands (with the extension **.CMD**) that guide the simulation process. This file consists of a list of instructions that are used to run the Functional Simulator in batch mode. (In interactive mode, these commands are entered in response to prompts displayed by the Functional Simulator, and the Command File is not required.)

The Functional Simulator uses these inputs to simulate the behavior of the design based on the input vectors. During this process, the Functional Simulator may be directed to create the following output files:

- (1) A Waveform Output File (with the extension **.WAV**) that contains a graphical waveform description of the nodes.
- (2) A Vector Table Output File (with the extension **.TBL**) that describes the state of the specified nodes.
- (3) A Log File (with the extension **.LOG**) that records the commands executed by the Functional Simulator. This file may be reused in future simulation runs.

- (4) A Save File (with the extension **.SAV**) that saves a simulation environment for future use.

You may choose one of two operating modes:

- Interactive mode
- Batch mode

In either mode, the Functional Simulator describes the logic of the design. In interactive mode, the simulation is performed step by step as you enter commands from the keyboard. You are able to break the simulation process, review its status, and then continue. In batch mode, on the other hand, you enter the input files, and the Functional Simulator performs the simulation and automatically sends the output to an output file. Afterwards, you are notified of the status of the simulation.

Figure FS-1 shows a block diagram of the Functional Simulator.

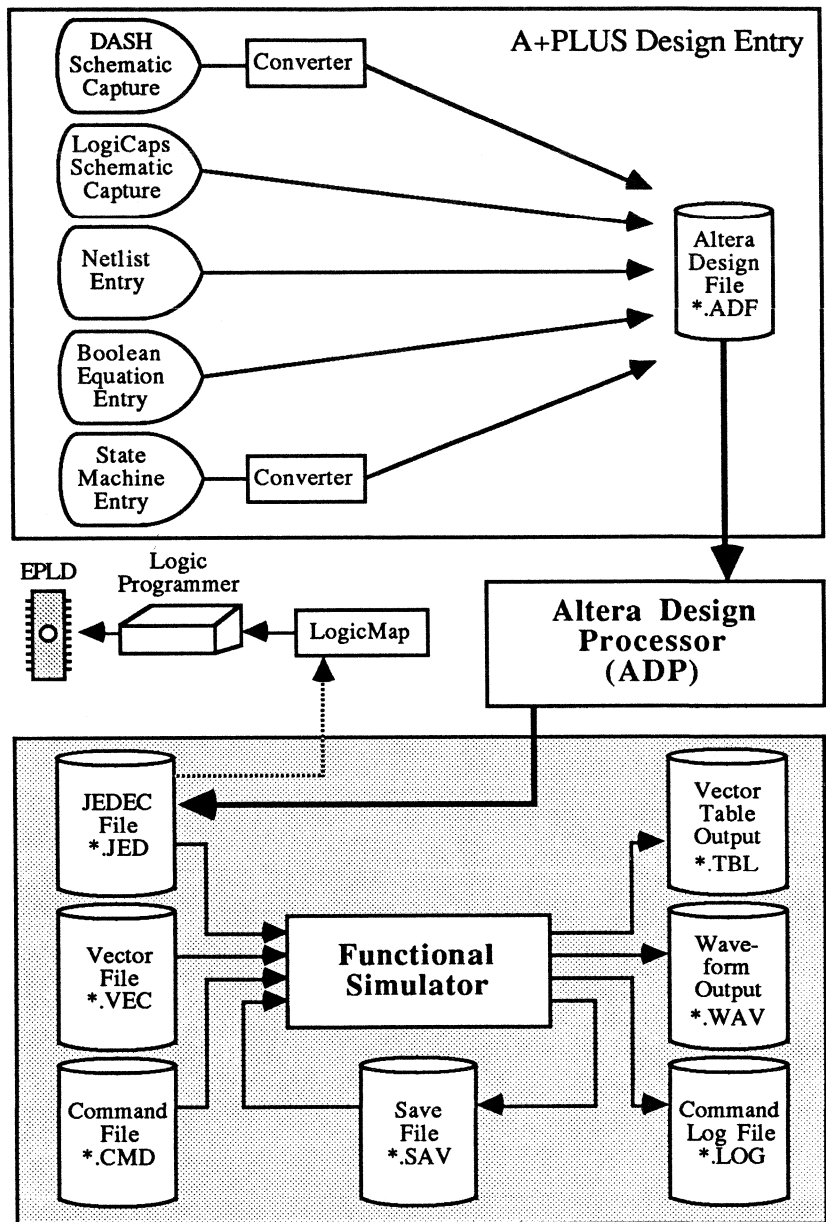


Figure FS-1. The Functional Simulator

The Simulation Process

Functional simulation occurs in three phases. In Phase 1, you set up the environment for the simulation. In Phase 2, you enter the desired simulation conditions (if any) and the commands that indicate what action the Functional Simulator should take when these conditions are met. In Phase 3, you enter the commands that display information on the simulation status and run the actual simulation.

Figure FS-2 illustrates the simulation process, including the commands that are appropriate for each phase. Detailed information on each command is available in the section tabbed *FSIM Reference* under *Command Reference*. Each simulation phase is described below.



For definitions of simulation terminology, refer to the section tabbed *FSIM Glossary*.

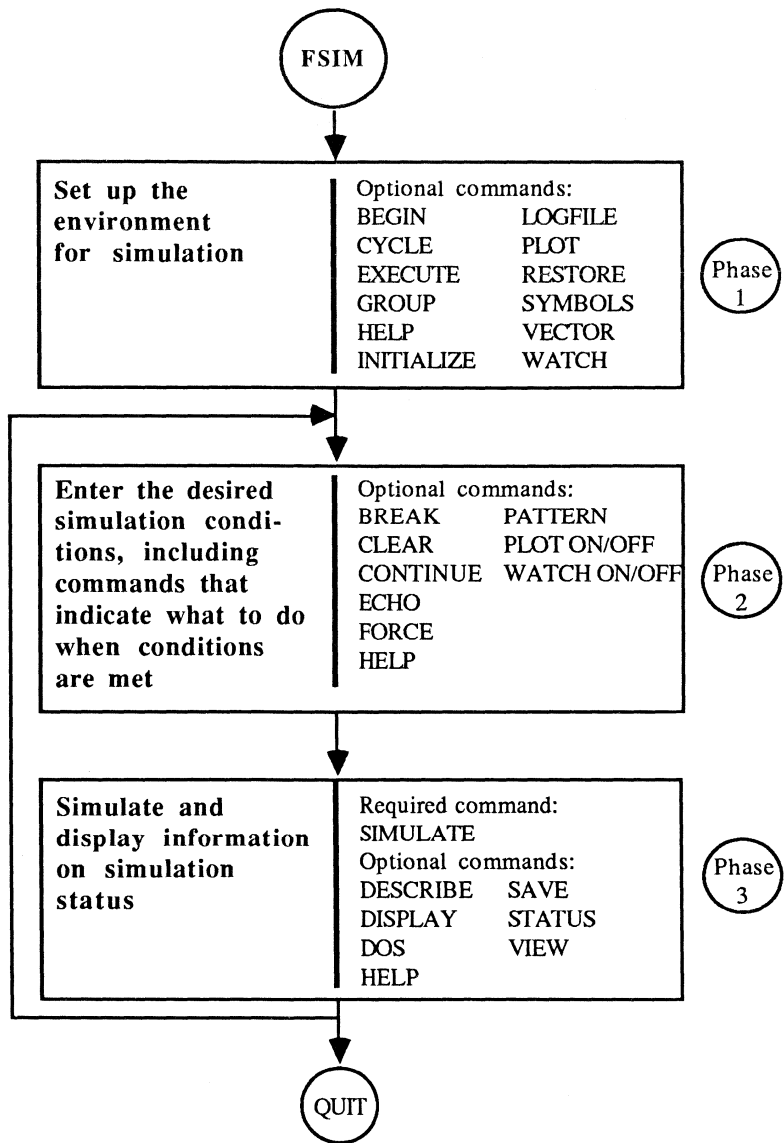


Figure FS-2. The Simulation Process

Phase 1

The first phase of the simulation process involves setting up the environment for the simulation, which is determined with the commands listed below. Once you have specified input vectors with the **VECTOR** or **PATTERN** command, all other Phase 1 commands are optional. (Remember that although the Functional Simulator symbolically tests the logic of your design, this representation is only as good as the input you provide by specifying test vectors.)

- The **VECTOR** command specifies the Vector file (**.VEC**) to be used for a particular simulation. (Your **.VEC** file(s) may be entered with a standard text editor.)
- As an alternative to using the **VECTOR** command, you may use the **PATTERN** command to enter vector patterns.
- The **CYCLE** command specifies the number of vector clocks in a cycle. (Refer to *Vector Processing and the CYCLE Command* in *FSIM Reference*.)
- The **INITIALIZE** command sets nodes to a specific level.
- The **GROUP** command groups nodes (for example, the elements of a bus) so that they may be addressed as a single group during the simulation. (Note: groups must be defined before you enter the **VECTOR** command.)
- The **SYMBOLS** command displays all node names being used.
- The **PLOT** command prints the simulation output as a waveform.
- The **WATCH** command prints the simulation output as a table.
- The **LOGFILE** command records all commands entered during simulation into a file. Later, you may use this file as the input to an **EXECUTE** command.
- The **RESTORE** command loads a previously saved simulation.
- The **BEGIN** command resets the Functional Simulator.
- The **HELP** command may be used at any time to display help information.

Phase 2

The second phase of the simulation involves entering the desired simulation conditions (if any) and the commands that indicate what action the Functional Simulator should take when these conditions are met. You may use any of the following commands (all Phase 2 commands are optional):

- The **BREAK** command allows you to set and check a variety of simulation conditions. For example, you may verify logic conditions in your design; check for faulty logic conditions; monitor the state of nodes during a certain time period; or change input vectors depending on the logic levels or cycle values reached during a simulation.
- Using breakpoints set with the **BREAK** command, the **PLOT** and **WATCH** commands allow you to output selected portions of the simulation in wave or table format, respectively.
- The **CONTINUE** command instructs the Functional Simulator to continue after it has evaluated a specific breakpoint and found the specified conditions to be true.
- The **CLEAR** command removes one or all set breakpoints.
- The **FORCE** command forces a certain node to a specific level. This command is similar to the **INITIALIZE** command, but while **INITIALIZE** may be used only at cycle 0, **FORCE** may be used at any cycle.
- The **ECHO** command allows you to write a text string to the terminal while you are executing a Command file in batch mode.
- The **HELP** command may be used at any time to display helpful information on the simulation commands.

Phase 3

The final phase of the simulation involves entering the commands that display information on the simulation status, and running the actual simulation.

After specifying the conditions for running the simulation, you must enter the **SIMULATE** command. If you do not specify a cycle value, the simulation will run for one cycle. Otherwise, you enter a decimal number indicating the highest cycle to which the simulation should run. All other Phase 3 commands are optional.



You may simulate to any cycle greater than the current cycle. For example, if you are at cycle 1, you may simulate to any cycle above 1; if you are at cycle 100, you may simulate to cycle 120 but not to cycle 90.

- The **DESCRIBE** command provides information about nodes and groups.
- The **STATUS** command displays information on the current status of the simulation.
- The **SAVE** command saves the current state of the simulation.
- The **DISPLAY** command displays the Log File, Plot File, Watch File, or Vector File.
- The **HELP** command displays helpful information on all simulation commands.
- The **VIEW** command invokes the Virtual Logic Analyzer and displays the results of the simulation in graphical waveform. You may enter the **VIEW** command to analyze your results at any time during or after simulation. (Refer to the section tabbed *Virtual Logic Analyzer*.)
- The **DOS** command temporarily interrupts the Functional Simulator and executes any command valid in DOS.
- **QUIT** exits the Functional Simulator and returns you to DOS or to A+PLUS, i.e., to the environment where it was invoked.

Simulator Inputs

The Functional Simulator uses three input files to process your design:

- JEDEC File (**.JED**) as generated by the Altera Design Processor
- Vector File (**.VEC**)—or vector patterns entered with the **PATTERN** command
- Command File (**.CMD**)—used only in batch mode

The input requirements for these files are described below.

JEDEC File Input

The JEDEC File generated by the Altera Design Processor (ADP) describes your design. A valid JEDEC File (which has the extension **.JED**) indicates that your design has no *syntactic* errors; simulation helps you determine whether your design contains any *logic* errors.

Vector File Input

The Vector File, which is entered with a standard text editor, describes the input waveforms on which the simulation is based. The input vectors define the simulation process by specifying logic levels of nodes in a design. Input levels drive the input pins and determine the internal logic levels. (Expected output levels are not definable.)

All input vectors are stored in vector files with the default extension **.VEC**. You may switch from one vector file to another during the simulation, although only one file can be active at any given time. The Functional Simulator reads the active vector file and builds a temporary file for the specified vectors.

Legal Logic Level Characters

The legal characters indicating different logic levels of vectors are as follows:

1	Logic high
0	Logic low
Z	High impedance (no input to pin); used for bidirectional pin when pin is used for output.
0–9, A–F	Used for groups with an assigned number base (binary, octal, hexadecimal, or decimal values).
X	Undefined

Legal Input Vector Format

Two formats—Pattern and Table formats—are available for specifying input conditions. You may use either format but not both in a single file.

To avoid repetition of lengthy or common patterns, you may group individual nodes and specify repeating patterns.



(1) To group individual nodes, you must use the **GROUP** command during Phase 1 of the simulation process *before* entering the **VECTOR** command.

(2) Only one vector table is allowed per file.

Pattern Format

The pattern format begins with the keyword **PATTERN:**, followed by a column of node names. Each node name is followed by the sequence of values specified for it. The inputs for a three-bit binary counter may therefore be expressed as follows:

```
PATTERN:  
nodec = 0 1 0 1 0 1 0 1 ;  
nodeb = 0 0 1 1 0 0 1 1 ;  
nodea = 0 0 0 0 1 1 1 1 ;
```

This syntax allows you to repeat all or part of an entire pattern. The repeat factor, indicated by an asterisk (*), is a decimal number showing how many times the pattern is to be repeated. The pattern to be repeated is enclosed in parentheses followed by the repeat factor. An * without a number indicates an infinitely repeating pattern (for example, (10)* = 101010 ...).

The node patterns of the three-bit counter shown above may therefore also be expressed as follows (note that nested patterns are also allowed):

```
nodec    = (01)*4 ;
nodeb    = (0011)*2 ;
or in a nested pattern: nodeb = ( (0)*2 (1)*2 )*2 ;
nodea    = (0)*4 (1)*4 ;
```

Note that (1) a space must be between the repeat factor and the next vector or vector pattern, and (2) infinite patterns may not be nested.



A pattern defined in the Vector File may be overridden with the **FORCE** or **PATTERN** command.

Table Format

The table format for input vectors begins with the keyword **TABLE:**, followed by a list of node levels and a column of node values. For example, inputs for a simple three-bit binary counter may be expressed as follows (each entry represents one column in the table):

```
TABLE: nodea nodeb nodec ;
      0      0      0
      0      0      1
      0      1      0
      0      1      1
      1      0      0
      1      0      1
      1      1      0
      1      1      1
```

In a group entry within a table, the column is interpreted according to the base specification for that group. The following example shows a vector table that contains individual nodes and a group represented with hexadecimal numbers:

```
TABLE: node1 node2 group3 ;
      0      1      5
      0      0      6
      0      0      7
```



A table may be overridden with the **FORCE** or **PATTERN** command.

Input vectors may also be specified with predefined vector sequences. Refer to *Predefined Input Vector Sequences* in the section tabbed *FSIM Reference*.

Command File Input

Command File input is used only for running a simulation in batch mode. This file contains the simulation instructions, and must have the extension **.CMD**. The commands specified determine the actions of the Functional Simulator in the same way as commands entered in response to the prompts displayed in the interactive mode. Functional Simulator commands are listed in alphabetical order and described in detail in *Simulation Commands*.

Simulator Outputs

The Functional Simulator may output a Log File (.LOG), Watch File (.TBL), Plot File (.WAV), and Save File (.SAV). Each type of output file is described below.

Log File

The optional command Log File (with the default extension .LOG) records the commands executed by the Functional Simulator. The Functional Simulator stops writing to the specified Log File when the **QUIT** or **LOG OFF** command is entered. This file may be used later to repeat the simulation session by renaming it (as a Command File) with the extension .CMD.

Watch File

The optional Watch File (with the default extension .TBL) is a vector table output file that provides a tabular description of the nodes requested by the **WATCH** command. The vector table output describes the state of the nodes in 1's and 0's, and may be printed at any time during or after simulation. (Refer to the description of the **WATCH** command in *Command Reference*.)

Plot File

The optional Plot File (with the default extension .WAV) is a waveform output file that contains a simple waveform description of individual nodes requested with the **PLOT** command. These waveforms may be printed at any time during or after simulation. (Refer to the description of the **PLOT** command in *Command Reference*.)

Both the Watch and Plot files include a line of the following format:

Simulation Cover : n %

where **n** indicates the percentage of nodes in the design that have had a transition from low to high or high to low during simulation. This feature informs you of how well the design has been exercised during a particular simulation run.

Save File

The Save File (**.SAV**) is an optional file that saves the simulation environment that exists when the **SAVE** command is entered. This file allows you to return to a particular simulation state and thus avoid repetition of simulation sessions. The simulation environment in the Save File may be restored at any time with the **RESTORE** command.

Sample Session — Interactive Mode

This sample session provides a step-by-step example of how to run a functional simulation, using the beverage dispenser design described in detail in *Boolean Equation Entry* in the **A+PLUS User Guide**. For quick reference, the **BEVDIS** circuit is shown here again in Figure FS-3. The Altera Design File for this sample is also provided on your **FSIM** diskette under the name **BEVDIS.ADF**.

This sample session generally follows the three-phase format of the simulation process described above and shown in Figure FS-2. Several commands from each phase are worked into the sample to illustrate their use.

To prepare for this sample session, copy the file **BEVDIS.ADF** and run it through the Altera Design Processor to produce the JEDEC file **BEVDIS.JED**. You may wish to create a directory named **BEVDIS** that will contain the files necessary for simulation, then change to that directory to perform simulation.

As you go through this sample simulation, refer also to *Command Reference* in the *FSIM Reference* section for detailed descriptions of all available commands.

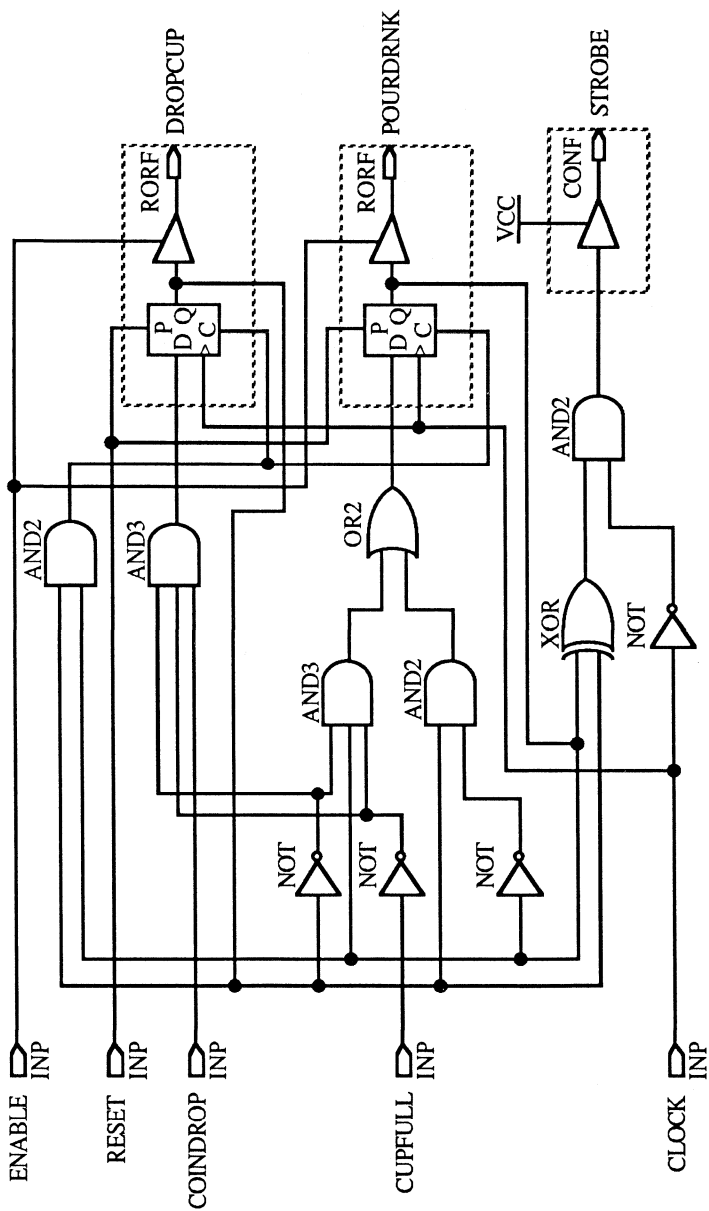


Figure FS-3. BEVDIS Sample Circuit

The sequence of steps required for interactive mode simulation is as follows:

Step 1: Manually draw the input waveforms for your design (optional).

Step 2: Enter the input vectors with a text editor and save the file with the extension **.VEC**.

Step 3: If you are in the APLUS Menu, press **<F6>** to invoke the Functional Simulator (FSIM). (Do not specify a filename.)

You may also invoke the Functional Simulator directly from DOS. Type at the system prompt:

FSIM <Enter>

Step 4: Enter the commands as you are prompted by the Functional Simulator.



You may enter the **VIEW** command to analyze your design at any time during or after simulation. (Refer to the section tabbed *Virtual Logic Analyzer*.)

Phase 1 — Setup Commands

In Phase 1, you set up the environment for the simulation. First, you should make a manual drawing of the input waveforms you expect the Functional Simulator to generate. Later, you can compare your drawing with the computer-generated waveform output. Next, you create a Vector File to set up the inputs for the simulation process.

Step 1 — Draw the Waveforms Manually:

Manually draw the design input waveforms for the beverage dispenser. An example is shown in Figure FS-4.

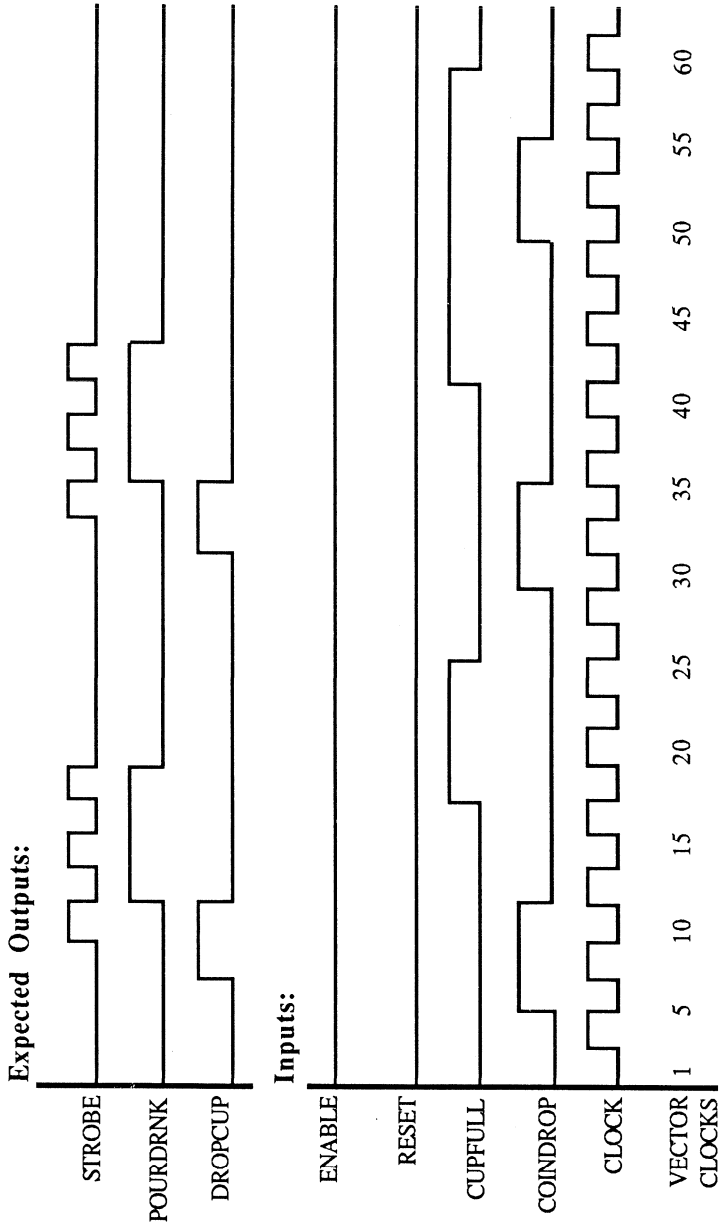


Figure FS-4. Manually Created Design Input and Output Waveforms

Step 2 — Create and Save the Vector File:

With your text processor in non-document mode, create a Vector File defining the set of input vectors. Save this Vector File with the name **BEVDIS.VEC**. Remember that if you wish to include a node group in your Vector File, the **GROUP** command must be entered before the **VECTOR** command. You may enter Patterns or Tables (not both), and only one table is permitted per file.

Pattern Format:

The pattern format begins with the keyword **PATTERN:**. This format is convenient for grouping repetitious node values. Figure FS-5 shows the vector file for the **BEVDIS** design. Comments are enclosed in percent symbols (%); patterns are separated by spaces.

```
PATTERN:

```

Figure FS-5. Pattern Format Vector File for BEVDIS

Table Format:

The tabular format begins with the keyword **TABLE:**. If you wish to simulate a large number of vector clocks, this format can become rather long. Figure FS-6 shows a portion of the **BEVDIS** pattern converted into tabular format. (Note: the remainder of this sample session uses the Pattern format.)

TABLE:
CLOCK COINDROP CUPFULL RESET ENABLE ;

0	0	0	0	1
0	0	0	0	1
1	0	0	0	1
1	0	0	0	1
0	1	0	0	1
0	1	0	0	1
1	1	0	0	1
1	1	0	0	1
0	1	0	0	1
0	1	0	0	1
1	0	0	0	1
1	0	0	0	1
0	0	0	0	1
0	0	0	0	1
1	0	0	0	1
1	0	0	0	1
0	0	1	0	1
0	0	1	0	1
1	0	1	0	1
1	0	1	0	1
0	0	0	0	1
0	0	0	0	1
1	0	0	0	1
1	0	0	0	1
0	0	0	0	1
0	0	0	0	1
1	0	0	0	1
1	0	0	0	1
0	1	1	0	1
0	1	1	0	1

Figure FS-6. Table Format Vector File for BEVDIS

Step 3 — Invoke the Functional Simulator:

To begin interactive mode simulation, you must now invoke the Functional Simulator.

While in the APLUS Menu, press <F6>. When you are prompted for a file to simulate, press <Enter>.

After the introductory information has been displayed, you are prompted for:

JEDEC File Name:

Type (without the extension):

BEVDIS <Enter>

The FSIM prompt symbol is displayed, with 0 indicating that you are at cycle 0:

0>



After you enter the **SIM** command, the **0** will change to the number indicating the current cycle.

Step 4 — Enter the Commands:

Enter all the commands to be executed during simulation (this example contains 13 commands). These commands determine the execution of the Functional Simulator as it processes the .VEC file during the actual device simulation run. The actual simulation will be performed only after you enter the **SIM** command. Each command is executed with <Enter>.

(1) Type:

0> LOG @bevdiss <Enter>

LOG tells the Functional Simulator to create a file containing all the commands entered during this session. The filename defaults to **BEVDIS.LOG**. Later, if you wish to repeat the same set of commands in batch mode, you may rename **BEVDIS.LOG** as **BEVDIS.CMD** and simply use the command **EXEC**.

(2) Type:

```
0> SYMB ???* <Enter>
```

The screen displays the list:

.P11	.P12	.P13	.P14
.P15	.P16	.P17	.P18
.P19	CLOCK	ENABLE	RESET
COINDROP	CUPFULL	DROPCUP	POURDRNK
STROBE			

This command and the pattern matching feature display a list of all node names containing three or more characters. (Node names of the format **.P#** are predefined node names for EPLD pins.)

(3) Type:

```
0> GROUP BINARY CONTROL = ENABLE RESET <Enter>
```

This command sets up a binary group (base 2) called **CONTROL** that contains **ENABLE** and **RESET**. Groups used in the Vector File must be entered with the **GROUP** command before the **VECTOR** command.

(4) Type:

```
0> VEC @bevdis <Enter>
```

This command tells the Functional Simulator to use the file **BEVDIS.VEC** for input vectors during **BEVDIS** simulation.

(5) Type:

```
0> GROUP HEX INPUTS = CLOCK CUPFULL COINDROP  
<Enter>
```

This command defines a hex group containing **CLOCK**, **CUPFULL**, and **COINDROP**. Note that this group has been entered *after* the **VECTOR** command; therefore, if a vector pattern was assigned to this group in the Vector File, it will be ignored.

(6) Type:

```
0> GROUP <Enter>
```

The screen displays the following:

Group CONTROL in binary contains the following members:
ENABLE RESET

Group INPUTS in hexadecimal contains the following members:
CLOCK CUPFULL COINDROP

The **GROUP** command displays the contents of all groups in this form.

(7) Type:

```
0> DESCRIBE CONTROL <Enter>
```

The screen displays the following:

Name	Pin	Macrocell	Level
CONTROL	-	-	X

The **DESCRIBE** command thus displays the level of the group labeled **CONTROL**.

(8) Type:

```
0> INIT CONTROL = 11 <Enter>
```

The **INIT** command forces the group to initialize at level 11 (binary). Note that in **BEVDIS.LOG** the **INIT** command is recorded as a **FORCE** command and the group is expanded to its individual members. (Refer to Figure FS-9).

(9) Type:

```
0> DESCRIBE CONTROL <Enter>
```

The screen displays the following:

Name	Pin	Macrocell	Level
CONTROL	-	-	11

The **DESCRIBE** command now displays the changed level of the **CONTROL** group.

(10) Type:

```
0> CYCLE 4 <Enter>
```

This command defines one cycle as equal to four vector clocks and the prompt will now increment every four vector clocks. (See *Vector Processing and the CYCLE Command in FSIM Reference*.)

(11) Type:

```
0> PLOT CLOCK ENABLE RESET COINDROP CUPFULL  
DROPCUP POURDRNK STROBE <Enter>
```

The **PLOT** command indicates which nodes should be included in the default waveform output file, **BEVDIS.WAV**.

(12) Type:

```
0> WATCH DROPCUP.FBK POURDRNK.FBK @watch.out  
<Enter>
```

The **WATCH** command indicates which nodes should be included in a tabular output file. The filename **WATCH.OUT** is specified.



Refer to *Referencing Subnodes* in the *FSIM Reference* section for instructions on adding an identifier to an internal node (i.e., subnode).

(13) Type:

```
0> PLOT OFF <Enter>
```

This command specifies that no waveform output will be generated until a **PLOT ON** command turns the waveform output on.

You now have specified the commands needed to set up the simulation environment. Next, you specify the simulation conditions.

Phase 2 — Control Commands

The **BREAK** command interrupts the simulation process so that one or more commands may be executed at a specified breakpoint. **BREAK** commands may be nested, but a nested **BREAK** command is only executed and a breakpoint set if the conditions for the root breakpoint have been met. When a list of breakpoint commands is entered, each **BREAK** command, as well as each command within the **BREAK** command, must be terminated with a semicolon (;).

(1) Type:

```
0> BREAK RANGE 5 DO PLOT ON; WATCH OFF; STATUS;  
BREAK RANGE 10 DO PLOT OFF; WATCH ON; STATUS;  
CONTINUE;; CONTINUE;; <Enter>
```

This nested command sequence causes the Functional Simulator to halt at cycle 5 and:

- start plotting a waveform output;
- stop generating tabular output;
- display the status of the simulation;
- set another breakpoint that will cause the simulator to halt at cycle 10 and:
 - stop plotting the waveform output;
 - start generating tabular output;
 - display the status of the simulation;
 - then continue the simulation without stopping;
- continue the simulation without stopping.

The screen will display the following:

```
[0] Range 5
```

where [0] is the number automatically assigned to the set breakpoint and **Range 5** indicates the halt conditions.

(2) Type:

```
0> BREAK RANGE 0 TO 10 NODES POURDRNK = 1  
CUPFULL = 1 <Enter>
```

The screen will display the following:

```
[1] Range 0 to 10 Nodes POURDRNK = 1 ..
```

where [1] is the number automatically assigned to the set breakpoint and **Range 0 to 10 Nodes POURDRNK = 1 ..** indicates the halt conditions.

This command causes the Functional Simulator to stop simulation if both **POURDRNK** and **CUPFULL** are high during the same cycle between cycles 0 and 10.

(3) Type:

0> BREAK <Enter>

The screen displays the state of the two breakpoints as follows:

[0] Range 5

[1] Range 0 to 10 Nodes POURDRNK = 1 ..

Next, the actual simulation is performed.

Phase 3 — Execution Commands

During this phase, the actual simulation is performed and information on the simulation status may be displayed.

(1) Type:

0> SIM 15 <Enter>

where **15** is the cycle to which the simulation should be performed.

The screen displays the following:

[1] hit at cycle = 4

indicating that the conditions set for breakpoint [1] have been met at cycle **4**. Therefore, the cycle number displayed before the prompt symbol (>) is now **4**.

(2) Type:

4> BREAK <Enter>

The screen displays the following:

[0] Range 5

[1] HIT Range 0 to 10 Nodes POURDRNK = 1 ..

The conditions for breakpoint [1] have been met.

(3) Type:

4> CLEAR 1 <Enter>

Breakpoint [1] is cleared.

(4) Type:

4> CONT <Enter>

Simulation will continue with the following display:

```
[0] hit at      cycle = 5  
Part in use   : EP310  
JEDEC file    : bevdis.jed  
Vector file   : bevdis.vec  
Logfile       : bevdis.log  
Logging       : ON  
Plot file     : bevdis.wav  
Plotting      : ON  
Watch file    : watch.out  
Watching      : OFF  
Sim. cover    : 81%  
[1]           Range 10  
[1] hit at    cycle = 10  
Part in use   : EP310  
JEDEC file    : bevdis.jed  
Vector file   : bevdis.vec  
Logfile       : bevdis.log  
Logging       : ON  
Plot file     : bevdis.wav  
Plotting      : OFF  
Watch file    : watch.out  
Watching      : ON  
Sim. cover    : 81%  
Simulation finished
```


This screen display shows that the conditions specified for breakpoint [1] have been met at cycle 5 and the commands have been executed. In addition, another breakpoint was set and hit, and its commands have been executed.

(5) Type:

15> QUIT <Enter>

You have now completed the sample session for interactive mode simulation. The figures on the following pages show:

- Your input to the keyboard (Figure FS-7)
- The output to the screen (Figure FS-8)
- The Log File **BEVDIS.LOG** (Figure FS-9)
- The Plot File **BEVDIS.WAV** (Figure FS-10)
- The Watch File **WATCH.OUT** (Figure FS-11)

```
LOG @bevdis <Enter>
SYMB ???* <Enter>
GROUP BINARY CONTROL = ENABLE RESET <Enter>
VEC @bevdis <Enter>
GROUP HEX INPUTS = CLOCK CUPFULL COINDROP <Enter>
GROUP <Enter>
DESCRIBE CONTROL <Enter>
INIT CONTROL = 11 <Enter>
DESCRIBE CONTROL <Enter>
CYCLE 4 <Enter>
PLOT CLOCK ENABLE RESET COINDROP CUPFULL DROPCUP POURDRNK
    STROBE <Enter>
WATCH DROPCUP.FBK POURDRNK.FBK @watch.out <Enter>
PLOT OFF <Enter>
BREAK RANGE 5 DO PLOT ON; WATCH OFF; STATUS; BREAK RANGE 10 DO
    PLOT OFF; WATCH ON; STATUS; CONTINUE;;CONTINUE; <Enter>
BREAK RANGE 0 TO 10 NODES POURDRNK=1 CUPFULL=1 <Enter>
BREAK <Enter>
SIM 15 <Enter>
CLEAR 1 <Enter>
CONT <Enter>
QUIT <Enter>
```

Figure FS-7. Input to the Keyboard

```

<Functional Simulator version information appears here>
.P11      .P12      .P13      .P14
.P15      .P16      .P17      .P18
.P19      CLOCK    ENABLE    RESET
COINDROP  CUPFULL   DROPCUP   POURDRNK
STROBE
Group CONTROL in binary contains the following members:
ENABLE    RESET
Group INPUTS in hexadecimal contains the following members:
CLOCK    CUPFULL   COINDROP
Name      Pin      Macrocell   Level
CONTROL   -        -           -           X
Name      Pin      Macrocell   Level
CONTROL   -        -           -           11
[0]
[1]      Range 5
[0]      Range 0 to 10 Nodes POURDRNK = 1..
[1]      Range 5
[1]      Range 0 to 10 Nodes POURDRNK = 1..
[1] hit at cycle = 4
[0]      Range 5
[1] HIT  Range 0 to 10 Nodes POURDRNK = 1..
[1] hit at cycle = 5
Part in use : EP310
JEDEC file : bevdis.jed
Vector file : bevdis.vec
Logfile    : bevdis.log
Logging    : ON
Plot file  : bevdis.wav
Plotting   : ON
Watch file : watch.out
Watching   : OFF
Sim. cover : 81%
[1]      Range 10
[1] hit at cycle = 10
Part in use : EP310
JEDEC file : bevdis.jed
Vector file : bevdis.vec
Logfile    : bevdis.log
Logging    : ON
Plot file  : bevdis.wav
Plotting   : OFF
Watch file : watch.out
Watching   : ON
Sim. cover : 81%

```

Figure FS-8. Screen Output

```

SYMBOLS .P11 .P12 .P13 .P14 .P15 .P16 .P17 .P18 .P19 CLOCK ENABLE
      RESET COINDROP CUPFULL DROPCUP POURDRNK STROBE ;
GROUP BINARY CONTROL = ENABLE RESET ;
VECTOR @bevdis.vec ;
GROUP HEXADECIMAL INPUTS = CLOCK CUPFULL COINDROP ;
GROUP ;
DESCRIBE CONTROL ;
FORCE RESET = 1 ENABLE = 1 ;
DESCRIBE CONTROL ;
CYCLE 4 ;
PLOT CLOCK ENABLE RESET COINDROP CUPFULL DROPCUP POURDRNK
      STROBE ;
WATCH DROPCUP.FBK POURDRNK.FBK @watch.out ;
PLOT OFF ;
BREAK
      RANGE 5
      DO
          PLOT ON ;
          WATCH OFF ;
          STATUS ;
          BREAK
          RANGE 10
          DO
              PLOT OFF ;
              WATCH ON ;
              STATUS ;
              CONTINUE ;
          ;
      CONTINUE ;
;
BREAK
      RANGE 0 TO 10
      NODES POURDRNK=1 CUPFULL=1 ;
BREAK ;
SIM 15 ;
BREAK ;
CLEAR 1 ;
CONTINUE ;
QUIT ;

```

Figure FS-9. Output to the Log File BEVDIS.LOG

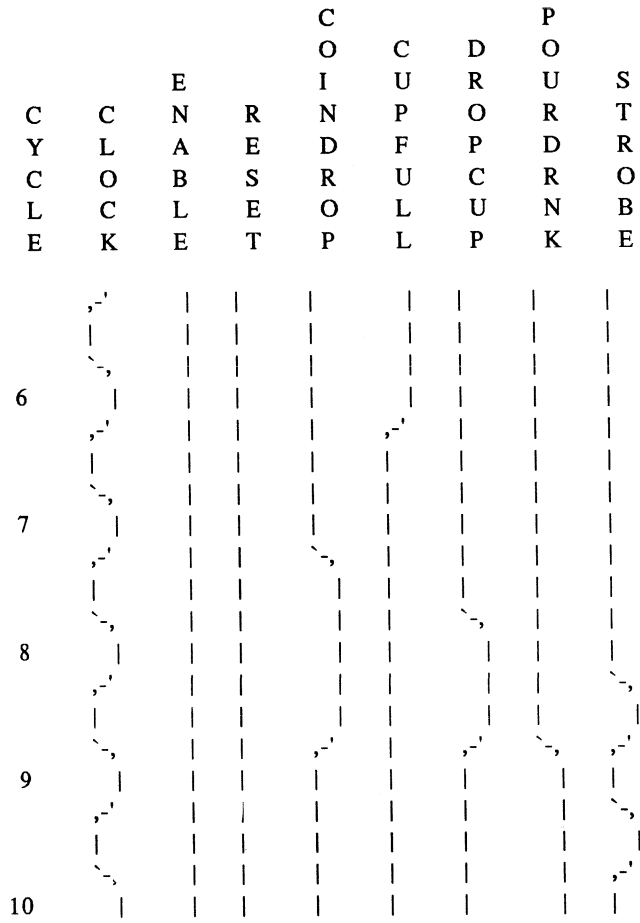


Figure FS-10. Output to the Plot File BEVDIS.WAV

		P
	D	O
	R	U
	O	R
	P	D
	C	R
	U	N
C	P	K
Y	.	.
C	F	F
L	B	B
E	K	K
	0	0
	0	0
	0	0
1	0	0
	0	0
	0	0
	1	0
2	1	0
	1	0
	1	0
	0	1
3	0	1
	0	1
	0	1
	0	1
4	0	1
	0	1
	0	1
	0	0
5	0	0
	0	1
	0	1
	0	0

Figure FS-11. Output to Watch File WATCH.OUT
(Part 1 of 2)

11	0	0
	0	0
	0	0
	0	0
12	0	0
	0	0
	0	0
	0	0
13	0	0
	0	0
	0	0
	0	0
14	0	0
	0	0
	0	0
	0	0
15	0	0

Simulation cover: 81%

**Figure FS-11. Output to Watch File WATCH.OUT
(Part 2 of 2)**

Sample Session — Batch Mode

Like the interactive mode sample session, this sample session uses the **BEVDIS** circuit shown in Figure FS-3. Before beginning batch mode simulation, you should copy **BEVDIS.ADF** from your **FSIM** diskette and run it through the Altera Design Processor to Processor to produce the JEDEC File **BEVDIS.JED**. You may wish to create a directory named **BEVDIS** that will contain the files necessary for simulation, then change to that directory to perform simulation.

As you go through this sample simulation, refer also to *Command Reference* in *FSIM Reference* for detailed descriptions of all available commands.

The sequence of steps required for batch mode simulation of the **BEVDIS** design are outlined below.

Step 1 — Draw the Waveforms:

Manually draw the input waveforms for the beverage dispenser (optional). Refer to the example in Figure FS-4.

Step 2 — Create and Save the Vector File:

Enter the input vectors with a text processor in non-document mode as described in *Interactive Mode (Step 2)* and save the file as **BEVDIS.VEC**.

Step 3 — Create and Save the Command File:

Type the Command File input with a text processor, as shown in Figure FS-12, and save it as **BEVDIS.CMD** (or copy **BEVDIS.LOG** from the interactive mode sample session to **BEVDIS.CMD**).

```

LOG @bevdis ;
SYMB ???* ;
GROUP BINARY CONTROL = ENABLE RESET ;
VEC @bevdis ;
GROUP HEX INPUTS = CLOCK CUPFULL COINDROP ;
GROUP ;
DESCRIBE CONTROL ;
INIT CONTROL = 11 ;
DESCRIBE CONTROL ;
CYCLE 4 ;
PLOT CLOCK ENABLE RESET COINDROP CUPFULL DROPCUP POURDRNK
    STROBE ;
WATCH DROPCUP.FBK POURDRNK.FBK @watch.out ;
PLOT OFF ;
BREAK RANGE 5 DO PLOT ON; WATCH OFF; STATUS; BREAK RANGE 10 DO
    PLOT OFF; WATCH ON; STATUS;
    CONTINUE;; CONTINUE;; ;
BREAK RANGE 0 TO 10 NODES POURDRNK=1 CUPFULL=1 ;
BREAK ;
SIM 15 ;
BREAK ;
CLEAR 1 ;
CONTINUE ;
QUIT ;

```

Figure FS-12. Command File BEVDIS.CMD for Batch Mode

Step 4 — Run the Functional Simulator in Batch Mode:

To begin batch mode simulation, you must now invoke the Functional Simulator.

While in the APLUS Menu, press <F6>. You may also invoke the Functional Simulator directly from DOS. Type at the system prompt:

FSIM <Enter>

When you are prompted for a file to simulate, press <Enter>.

After the introductory information has been displayed, you are prompted for:

JEDEC File Name:

Type (without the extension):

BEVDIS <Enter>

The Functional Simulator will look for the three input files (i.e., **BEVDIS.JED**, **BEVDIS.VEC**, and **BEVDIS.CMD**) and process them. After the simulation has been completed, the message:

Simulation finished

is displayed.

This concludes the sample session for batch mode simulation.

FSIM Reference

This section explains some advanced features/concepts used with the Functional Simulator and provides detailed descriptions of all commands. (Refer to *Appendix C* in the **A+PLUS Reference Guide** for a description of the BNF syntax specifications used.)

You will find information on:

- Vector processing and the **CYCLE** command
- Using predefined input vector sequences
- Propagation of undefined node levels
- Simulating designs with bidirectional pins
- Simulating designs with buses
- Simulating designs with externally connected clock pins
- Referencing predefined node names for pins, macrocells, and buses
- Referencing subnodes of I/O primitives
- Reserved words not available for use with FSIM
- FSIM command format
- Descriptions of syntax and usage of each command

Vector Processing and the CYCLE Command

The sequence of events for processing a vector in a Vector File is given below. This sequence is executed at each vector clock, i.e., in the unit of time during which a single vector is processed.

- (a) A vector is loaded and input pin values are updated.

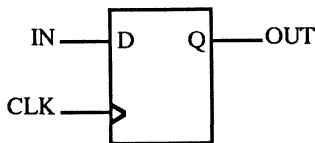


FORCE commands always override vector inputs.

- (b) If a clock pin goes high as a result of (a), the synchronous register outputs are updated. If the part specified in the JEDEC File is an EP310, the synchronous Preset product term is evaluated and may also cause the register output to be updated.
- (c) If the level of a pin-driven input (e.g., Clock, Read Strobe, or Write Strobe) to a BUSTER (EPB1400) latch or BUSX changes as a result of (a), the latch or transceiver is reevaluated.
- (d) If an asynchronous clock product term goes high as a result of (a), the register output is updated.
- (e) If a Clear (or Output Enable) product term goes high as the result of (a), the register output (or pin level) is updated.
- (f) Other product terms in the array are now evaluated.
- (g) BUSTER latches and bus transceivers are reevaluated.
- (h) If any feedbacks have changed during this process, steps (d) through (g) are repeated until the feedback levels are stable or the convergence limit is reached.

This order of processing may cause unexpected results. Consider the following example:

You have a D flipflop:

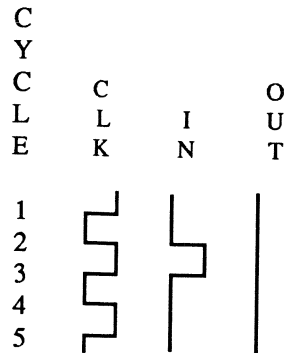


This flipflop is fed a Vector File containing the following:

```

CLK    = 0 1 0 1 0 ;
IN     = 0 1 0 0 0 ;
  
```

The resulting waveform is as follows:



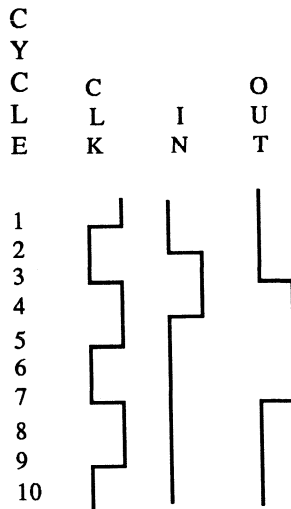
The IN was not passed through to OUT at vector clock 2 because the synchronous register clock was processed before the input. It is possible to avoid this situation by manually expanding the length of all signals and shifting signals relative to each other.

The expanded Vector File would appear as follows:

```

CLK = 0 0 1 1 0 0 1 1 0 0 ;
IN  = 0 1 1 0 0 0 0 0 0 0 ;
  
```

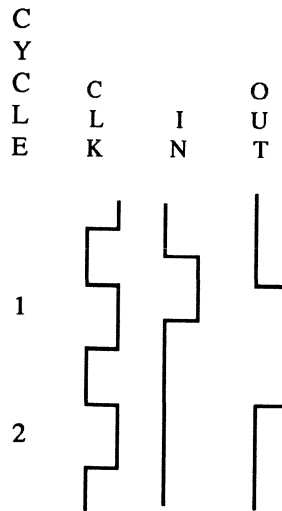
The waveform output is "stretched out," as follows:



The relative length of all signals is maintained; yet the circuit now simulates as expected.

The **CYCLE** command is provided to simplify the use of the Functional Simulator and interpretation of the output. This command instructs the Functional Simulator to display and use all values relating to vector clocks in terms of a user-specified cycle. This cycle is an integral number of vector clocks.

If you specify a cycle of 4 with the command **CYCLE 4**, the waveform output of the example would appear as follows:



The labeling of the waveform now corresponds to the cycles of CLK.



The **CYCLE** command has no effect on the simulation process or on how the Vector File is read. It is simply a means of simplifying the interpretation of the output waveforms.

When the cycle length is changed, only the following areas are affected:

- Input cycle values for **SIM** and **BREAK** commands
- Output cycle value of the prompt
- Output labeling for Plot and Watch files
- Predefined vector input sequences (see below)

Predefined Input Vector Sequences

Four predefined vector sequences, which enable you to define vectors for individual groups, are available. These sequences are:

- Binary Counting (**COUNT**) Sequence
- Rotating Bit (**ROTATE**) Sequence
- Gray Code (**GRAY**) Sequence
- Glitch Generator (**GLITCH**) Sequence

Binary Counting Sequence

If **n** members are in the group, the **COUNT** sequence begins with **0** and counts modulo 2^n . For a group with three members, the sequence counts from **0** to **7** and then begins again with **0**:

000 001 010 011 100 101 110 111 000 001 ...

Example:

```
grp1 = COUNT ;
```

Rotating Bit Sequence

In the **ROTATE** sequence, a bit with level **1** is rotated left through each bit position within the group. Startup is with the right-most bit position set, and the sequence repeats every **n** cycles for an **n**-member group. Therefore, the rotating bit pattern for a three-member group is:

001 010 100 001 010 ...

Example:

```
grp1 = ROTATE ;
```


Gray Code Sequence

The **GRAY** sequence changes one bit at a time. In a group with **n** members (bits), the sequence has 2^n unique elements in the sequence. These are identical to the elements of the **COUNT** sequence, but in a different order:

000 001 011 010 110 111 101 100 000 001 011 ...

Example:

grp1 = GRAY ;

Glitch Generator Sequence

The **GLITCH** sequence changes the maximum number of bits in a pattern from one cycle to the next. It is like the **GRAY** sequence, but the bits are inverted in every other pattern of the sequence:

000 110 011 101 110 000 101 011 000 110 011 ...

Example:

grp1 = GLITCH ;



The cycle value affects the rate at which predefined input vector sequences change. For example, if **CYCLE** is 4, the **COUNT** sequence increases every four vector clocks.

Propagation of Undefined Node Levels

If your design contains nodes with undefined logic levels, they will propagate these according to the rules given below.

Logic Functions

If at all possible, defined levels are maintained. For example, if a logic low (0) is ANDed with an undefined level, the result is a logic low. The following levels are propagated for the AND, OR, and NOT logic functions (0 = logic low, 1 = logic high, X = undefined):

AND Function

0	AND	X	=	0
1	AND	X	=	X
X	AND	X	=	X

OR Function

0	OR	X	=	X
1	OR	X	=	1
X	OR	X	=	X

NOT Function

NOT	X	=	X
-----	---	---	---

Flipflops and Latches

- (1) If the output level of a flipflop or latch would remain constant regardless of an undefined input level, this output level is maintained. For example, if an undefined asynchronous clear level is applied to a flipflop with an output at logic low, the output would be maintained at logic low (0). Or, if an undefined clock is applied to a D flipflop with the D node at logic high (1) and the output Q node at logic high, a logic high is maintained on the Q node.
- (2) If the output level of a flipflop or latch would change depending on an undefined input level, the output node is set to undefined (X) level.

Output Enables

If a tri-state buffer has an undefined Output Enable input, the output of that buffer will be undefined (X).

Bidirectional Pins

If you wish to simulate a design that uses bidirectional I/O pins, three extra steps are required to set up the environment for simulation and view the results for the pins:

1. Set Output Enable (OE) to 0 (low) one vector clock *before* the pin is used for input and to 1 (high) one vector clock *before* it is used for output. This allows the OE signal to propagate through to the tri-state buffer before input or output begins.
2. When specifying vector patterns, use Z (high impedance) when the pin is driving out.

Example: For a design with a bidirectional pin IOPIN and Output Enable input pin OEPIN, you enter:

```
0> PAT OEPIN = 1 1 1 0 0 0 0 1 1 1 1
0> PAT IOPIN = Z Z Z Z 0 1 0 1 Z Z Z
                % Z's when pin is driving out %
```

3. When you plot or print the nodes (with PLOT or WATCH), append the extension .INP to the bidirectional pin name in order to view the input values. (If the .INP extension is not used, Z's will be plotted instead.)

For the example above, the command

```
0> WATCH OEPIN IOPIN IOPIN.INP
```

yields the following output (note that IOPIN values for cycles 1-4 and 9-11 are defined by output logic not described in this example):

	C	O	I	I
	Y	E	O	O
	C	P	P	P
	L	I	I	I
	E	N	N	N
	1	1	0	Z
	2	1	1	Z
	3	1	0	Z
	4	0	1	Z
	5	0	Z	0
	6	0	Z	1
	7	0	Z	0
	8	1	Z	1
	9	1	0	Z
	10	1	1	Z
	11	1	0	Z

Bus Port Pins

If you wish to simulate bus operation in a BUSTER (EPB1400) design, two additional steps are required to set up the environment for simulation and view the results for the bus port pins, *except* when the control for the bus transceiver (BUSX) is supplied *only* by the dedicated Read Strobe pin (/CRS). This procedure applies to any other combination of up to three of the following inputs: Read Strobe (/RS), Read Enable (RE), and Output Enable (OE).

1. Set the signals that control BUSX so that the bus port is driving *in* one vector clock *before* the bus port pin is used for input, i.e., RS = 0, OE = 1, RE = 0; and so that it is driving *out* one vector clock *before* the bus port pin is used for output, i.e., RS = 0, OE = 1, RE = 1.
2. When specifying vector patterns, use Z (high impedance) when the bus port is driving *out*.

Externally Connected Clock Pins

For some designs, the Utilization Report (.RPT file) contains a message to connect clock pins externally. To simulate your design, you must use predefined node names to reference each of the clock pins specified in the Utilization Report. In addition, you must define the same input vectors for each clock pin.

For example, if your EP600 design requires that you externally connect pins 1 and 13, your Vector File input for simulation must contain lines of the following format:

```
PATTERN:  
.P1   = (0 1)*4 ;  
.P13  = (0 1)*4 ;
```

Referencing Predefined Node Names

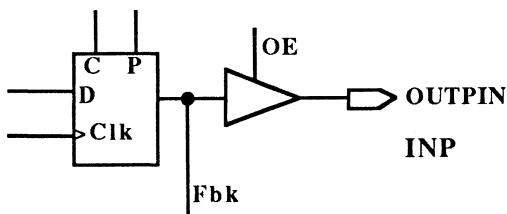
To allow complete access to all nodes within an EPLD, the Functional Simulator automatically creates node names for each element of BUSTER (EPB1400) buses, pins (except VCC and GND) and macrocells:

- Each element of a BUSTER bus may be referenced as **.B0** through **.B7**, where **.B0** and **.B7** are the least and most significant bits, respectively.
- Pins may be referenced as **.P#**, where # represents a 1- or 2-digit number. (EP1800G pin numbers also include a letter, e.g., **.PA10**.)
- Macrocells may be referenced with **.M#**, where # represents a 1- or 2-digit number.

These predefined names may be used in the same manner as user-defined node names and displayed with the **SYMBOLS** command.

Referencing Subnodes

Individual pin nodes may be referenced by their pin names, whereas subnodes (i.e., internal nodes) of I/O primitives (namely, Output Enable (OE), Clear (C), Preset (P), Clock (Clk), Feedback (Fbk), and Input Vector (INP)) are referenced by appending an identifier to the pin name. For example, the following illustration shows a generic representation of an I/O primitive where the output pin name associated with the primitive is **OUTPIN**:



Subnodes of the I/O primitive are referenced with the following notation (both uppercase and lowercase are allowed and are significant):

OE	OUTPIN.OE
Clear	OUTPIN.C
Preset	OUTPIN.P
Clock	OUTPIN.CLK
Feedback	OUTPIN.FBK
D input	OUTPIN.D
T input	OUTPIN.T
Input Vector	OUTPIN.INP



The Functional Simulator will ignore any other types of node names, including user-assigned node names, that contain a period.

When a design passes through the Altera Design Processor, all JK and SR flipflops are emulated by D and T flipflops. Consequently, you cannot reference JK or SR flipflops directly, and J, K, S, and R subnode extensions are not allowed. (For description of JK and SR flipflop configurations, see *Altera Applications Brief AB28*.)

Reserved Words

The following list of words may not be used for node or group names.

BEGIN	PLOT
BIN	QUIT
BINARY	RA
BREAK	RANGE
CLEAR	ROTATE
CONT	REST
CONTINUE	RESTORE
COUNT	SAVE
CYCLE	SIM
DEC	SIMULATE
DECIMAL	SYMB
DESC	SYMBOLS
DESCRIBE	STATUS
DISP	TO
DISPLAY	VEC
DO	VECTOR
DOS	VIEW
ECHO	WATCH
EXEC	
EXECUTE	
FORCE	
GRAY	
GLITCH	
GROUP	
HELP	
HEX	
HEXADECIMAL	
INIT	
INITIALIZE	
LOG	
LOGFILE	
NO	
NODES	
OCT	
OCTAL	
OFF	
ON	
PAT	
PATTERN	

Command Reference

This section lists all available simulation commands in alphabetical order. Each command description is presented on a separate page and includes detailed information as outlined here:

Format: Gives the name and accepted abbreviation of the command. (Refer to *Command Format* below.) The format shows each command terminated by a semicolon, i.e., as entered in batch mode simulation. In interactive mode, however, the semicolon is optional.

Purpose: Explains the intended function of the command.

Description: Describes the function of the command.

Example: Shows an example of how the command is used.

Note: Gives additional information if needed.



1. Commands in a breakpoint command list, i.e., beginning with the **BREAK** command, must be terminated with a semicolon in both interactive and batch modes.

2. The at-symbol (@) indicates an (optional) pathname followed by a filename.

Command Format

The general format for simulation commands is as follows:

command <argument list> ;

where <argument list> consists of one or more of the following:

- a node list
- a node value list
- a cycle value
- a command list
- a pathname
- a reserved word
- a decimal number(s)

Node List

A node list consists of node and group names separated by white space. Example:

node1 node2 node3 node4 ...noden

The Functional Simulator supports pattern matching to simplify entry of node lists. Examples:

- E*** matches all node names that start with **E**, regardless of the number of characters in the name (8 is the maximum permitted).
- E?E** matches all node names that are 3 letters long, start with **E**, and end with **E**.
- *E** matches all node names that end with **E**.
- E??** matches all node names that are 3 letters long and start with **E**.

Node Value List

A node value list contains a node or group and the corresponding value for that node or group. Example:

node1 = value1 node2 = value2 node3 = value3

Cycle Value

The cycle value specifies the timing of the execution of a command with respect to the current cycle. The cycle value argument may be used with the **SIM** and **BREAK** commands. Example:

123 (absolute value)
+22 (relative value)

The relative value, indicated by the + symbol, is relative to the current cycle value. (Both examples are equivalent if the current cycle is 101.) Refer also to *Phase 3 of The Simulation Process*.

Command List

A command list is a set of commands, each of which is terminated by a semicolon. The **BREAK** command is the only command that may contain a nested list of commands. Example:

```
BREAK RANGE +22 DO FORCE NODE1 = 1; STATUS;  
PLOT ON; CONTINUE ;;
```

Commands in a list are separated by semicolons (;). The **BREAK** command list may not contain the **SIM**, **PATTERN**, or **QUIT** command.



In interactive mode, commands are executed by pressing <Enter>, i.e., they may not span lines; in batch mode, all commands are terminated with a semicolon (;).

Pathname

The pathname consists of an at-symbol (@), followed by a pathname (optional) and filename. Example:

```
@\FSIM\DOC\BEVDIS.CMD
```

Table FS-1 lists the commands according to their general function.

Table FS-1. Simulation Commands

FUNCTION	COMMAND
<p>Node commands Set signals to specified logic levels.</p>	<p>FORCE INITIALIZE (INIT)</p>
<p>Information commands Provide data about commands, nodes, and groups.</p>	<p>DESCRIBE (DESC) DOS GROUP HELP STATUS SYMBOLS (SYMB)</p>
<p>Simulation control commands Control execution of the simulation.</p>	<p>BREAK CLEAR CONTINUE (CONT) ECHO QUIT SIMULATE (SIM)</p>
<p>Input commands Specify input to the simulator.</p>	<p>BEGIN EXECUTE (EXEC) PATTERN (PAT) RESTORE (REST) SAVE VECTOR (VEC)</p>
<p>Output commands Specify the output format and display.</p>	<p>CYCLE DISPLAY (DISP) LOGFILE (LOG) PATTERN (PAT) PLOT VIEW WATCH</p>

BEGIN

Format: **'BEGIN' ;**

Purpose: Resets the Functional Simulator.

Description: Resets the Functional Simulator to its initial state. The current design and simulation environment are lost. All breakpoints and vectors are cleared; all groups are removed; and the current cycle is set to 0.

Example: **BEGIN ;**

BREAK

Format: **'BREAK'** [<breakpoint id> | ['**RANGE'**
<cycle value> ['**TO'** <cycle value>]] ['**NODES'**
<node value list>] ['**DO'** <command list>] ;

where:

<breakpoint id> is a decimal number identifying a breakpoint.

RANGE <cycle value> **TO** <cycle value> identifies a range of cycles that cause the Functional Simulator to stop when any node value list conditions are met.

NODES <node value list> identifies the values for the nodes that cause the Functional Simulator to stop when any cycle range conditions are met.

DO <command list> identifies the list of commands to be executed when the Functional Simulator stops when the breakpoint conditions have been met. Each command in the list must be terminated with a semicolon (in interactive or batch mode). For example, node logic levels may be changed at a particular breakpoint by using the **VEC** or **FORCE** command in this command list.

Purpose: Interrupts a simulation at the specified cycle and/or node values. During the break, you may execute a list of commands or display breakpoint status. This command is optional.

Description: When a breakpoint is encountered during the simulation, the Functional Simulator stops and the command list is executed. The command list may contain any command, including **BREAK**, except **SIM**, **PATTERN**, or **QUIT**. If the **BREAK** command is followed only by a breakpoint number, the state of the specified breakpoint is displayed; if it is used without

arguments, the **BREAK** command displays the state of all breakpoints.

Examples: **BREAK ;** (displays all breakpoints)
BREAK 22 ; (displays breakpoint 22)
BREAK RANGE +10 TO +30 NODES CLOCK =
1 OUTPUT1 = 0 DO DESCRIBE OUTPUT2;
STATUS; CONTINUE ; ;

CLEAR

Format: **'CLEAR' [<breakpoint id>] ;**

Purpose: Clears the specified breakpoint. If no breakpoint number is specified, all breakpoints are cleared. This command is optional.

Description: A breakpoint must be set before it can be cleared.

Examples: **CLEAR ;** (clears all breakpoints)
CLEAR 20 ; (clears breakpoint 20)

CONTINUE (CONT)

Format: 'CONT' ;

Purpose: Resumes simulation after a breakpoint has been encountered and the conditions specified in the **BREAK** command have been met. This command is optional.

Description: This command is valid only when the current cycle is less than the cycle specified in the most recent **SIM** command or until another breakpoint is encountered.

Example: **CONTINUE** ;

CYCLE

Format: '**CYCLE**' [<decimal number>];

Purpose: Specifies the cycle value. This command is optional.

Description: This command allows you to specify the cycle length in vector clocks. This cycle length is then used for input and output references to time. The default cycle length is one vector clock. Refer also to *Vector Processing and the CYCLE Command*.

The cycle value affects the rate at which predefined input vector sequences change. For example, if **CYCLE** is 4, the **COUNT** sequence increases every four vector clocks.

Examples: **CYCLE** ; (increments the cycle number at every vector clock)

CYCLE 2 ; (increments the cycle number every second vector clock)

DESCRIBE (DESC)

- Format:** 'DESC' <node list> ;
- Purpose:** Provides information about nodes and groups specified in the node list. This command is optional.
- Description:** This command provides information about specified nodes or groups. Information about nodes includes the current level and associated pin and macrocell numbers. **DESC** has no default node list: when the node list is empty, nothing is displayed when the command is invoked. This command has no effect on the state of the simulation.
- Examples:** **DESC DROPCUP COINDROP ;**
DESC P? ;
- Note:** Pattern matching with the wildcard characters ? and * may be used with this command. For example, if you enter **DESC E***, all nodes whose name starts with **E** are described, regardless of the number of characters in the name.

DISPLAY (DISP)

- Format: **'DISP' 'LOG' | 'PLOT' | 'WATCH' | 'VECTOR' | @<pathname> ;**
- Purpose: Displays a Log File, Plot File, Watch File, Vector File, or any ASCII file. This command is optional.
- Description: This command will display output files only for the currently simulated design. If **@<pathname>** is specified, it will display any ASCII file. (Use the **STATUS** command to determine whether the **PLOT**, **WATCH**, and **LOG** commands are turned on or off.)
- Examples: **DISP LOG ;** (displays the current Log File)
DISP @bevdis.adf ; (displays bevdis.adf)
- Note: **DISP** can only display a Log, Plot, or Watch File after the **LOG ON**, **PLOT ON**, or **WATCH ON** command has been used to create the file. Similarly, a Vector File can only be displayed after the **VEC** command has been used to specify the input Vector File to be used for simulation.

DOS

Format: **'DOS' <DOS command> ;**

Purpose: Executes a single DOS command.

Description: Temporarily interrupts the Functional Simulator to enable you to execute any DOS command. You are immediately returned to the Functional Simulator after the command is completed.

Examples: **DOS dir *.JED ;**
DOS edit <filename> ;

Note: To avoid memory allocation problems, do not use this command to execute "Terminate and Stay Resident" programs, e.g., DOS PRINT, GRAPHICS, SIDEKICK.

ECHO

Format: **'ECHO' <text> ;**

Purpose: Writes a text string to the terminal.

Description: This command is most useful while you are executing a Command File (.CMD) in batch mode. It enables you to display messages to the terminal as the file executes.

Examples: **ECHO display logfile ;**
ECHO Hello, how are you? ;

EXECUTE (EXEC)

- Format: **'EXECUTE' [@<pathname>] ;**
- Purpose: Specifies a Command File for immediate execution. The default is <JEDEC filename>.CMD. This command is optional.
- Description: **EXECUTE** informs the Functional Simulator to take command input from the Command File specified with **@<pathname>**. If this file contains a **QUIT** command, the Functional Simulator reads the file up to the **QUIT** command, closes it, and then returns to take command input from the preceding file or the keyboard. (Any number of nesting levels is allowed.) If the file does not contain a **QUIT** command, the Functional Simulator reads the entire file and then treats the End-of-File (EOF) as an implicit **QUIT** command.
- Example: **EXEC @INIT.CMD ;**

FORCE

- Format:** `'FORCE' <node value list> ;`
- Purpose:** Forces a node, nodes, or a group to a specified value regardless of the vector input at the current cycle. This command is optional.
- Description:** This command forces nodes or groups in the accompanying node value list to assume the level specified in the list. This command executes before the next vector is processed and may be used at any cycle.
- Examples:**
- ```
FORCE nodea = 1 grp1 = 55 ;
 (forces nodea high and grp1 to level 55 at the
 next vector clock)
FORCE RESET = 1 ;
 (forces RESET high at the next vector clock)
```
- Notes:**
1. The **FORCE** command sets a node to a user-specified level, regardless of its current level. This command is useful if you want to place specific nodes into a certain state to observe the results of subsequent simulation steps.
  2. The **FORCE** command is often used within a breakpoint command list. For example:  
  

```
BREAK NODES nodeA = 0 DO FORCE RESET
= 1;;
```

  
causes FSIM to force **RESET** high when node **nodeA** is low.

# GROUP

Format:           '**GROUP**' [[<base>] <group name> ['=' <node list>]] ;

Purpose:           Identifies the members of a <node list> as one group so that they may be referenced, processed, and displayed as <group name>. This command is optional.

Description:     The <base> for a group name may be binary (**BIN** –the default), hexadecimal (**HEX**), decimal (**DEC**), or octal (**OCT**). When used without arguments, the **GROUP** command displays the members of all groups. When used with a group name, this command displays the members of the group.

Notes:           1.    The first member of the node list is the most significant bit of the group value, the last member is the least significant.

                  2.    Node groups cannot be plotted; you may only plot the individual members of the group (i.e., use individual node names in the **PLOT** command node list.)

                  3.    You may group the individual nodes of a BUSTER (EPB1400) internal bus by using the predefined node names **.B0** through **.B7**.

Examples:       **GROUP HEX OUTPUT = Q\* ;**  
                  (puts all node names beginning with Q into the hexadecimal **OUTPUT** group)  
                  **GROUP INPUT ;**  
                  (displays the members of the **INPUT** group)  
                  **GROUP ;**  
                  (displays the members of all groups)



# HELP

Format: **'HELP'** [<command name>] ;

Purpose: Provides helpful information about a specific command. When used without a specific command name, it displays a list of all available commands. This command is optional.

Description: **HELP** prints a summary of information on each command in the command list. The information is automatically paged if necessary. At the **--More--** prompt, press **q** to quit help or press any other key to read the next page. **HELP** has no effect on the status of the simulation.

Example: **HELP BREAK ;** (displays help info about the **BREAK** command)

## INITIALIZE (INIT)

Format: **'INIT' <node value list> ;**

Purpose: Sets nodes or groups to the given value before simulation is started, i.e., at cycle 0. This command is optional.

Description: The **INIT** command forces nodes or groups in the accompanying node value list to assume the level specified in the list. This command executes before the next cycle is processed.

Examples: **INIT nodea = 1 grp1 = 55 ;**  
(initializes **nodea** high and **grp1** at level 55)  
**INIT RESET = 1 ;**  
(initializes **RESET** high)

Notes:

1. The **INIT** and **FORCE** commands set a node to a user-specified level, regardless of its current level. These commands are useful if you want to place specific nodes into a certain state to observe the results of subsequent simulation steps.
2. The **INIT** command may be used only at cycle 0. **FORCE** may be used at any cycle.

## LOGFILE (LOG)

- Format:** 'LOG' 'ON' | 'OFF' | [ @<pathname> ] ;
- Purpose:** Instructs the Functional Simulator to start or stop logging simulation commands to the specified or default Log File. This command is optional.
- Description:** This command turns logging on or off. The default Log File is <JEDEC filename>.LOG; a different Log File may be specified with @<pathname>.
- Examples:**
- LOG OFF ;** (turns logging off)
  - LOG ON ;** (turns logging on)
  - LOG ;** (turns logging on with output sent to <JEDEC filename>. LOG)
  - LOG @sim.log;** (turns logging on with output sent to sim.log)
- Note:** Any existing Log File at the default or specified <pathname> will be overwritten.



# PLOT

Format: **'PLOT' 'ON' | 'OFF' | <node list> [*@<pathname>*] ;**

Purpose: Turns the plotting function on or off. **PLOT** command output consists of a printable ASCII file containing a waveform representation of the node levels in the node list. The default Plot File is **<JEDEC filename>.WAV**; a different Plot File may be specified with **@<pathname>**. This command is optional.

Description: **PLOT** output consists of characters that describe a waveform for each node in the **<node list>**. Groups may not be used in the **<node list>**; instead, you must plot the individual members of the group. The output runs horizontally across the paper, so that the waveforms can continue over multiple pages. The characters used to display the level of the nodes in the Plot File are as follows:

|                        | <b>L H</b> |
|------------------------|------------|
| High level             |            |
| Low level              |            |
| Low to High transition | , -'       |
| High to Low transition | ' - ,      |
| High-Z (tri-state)     | Z          |
| Undefined              | X          |

**PLOT** also sets up the node list that specifies the nodes monitored by the Virtual Logic Analyzer. Refer to the section tabbed *Virtual Logic Analyzer*.

Like **WATCH** and **STATUS**, **PLOT** displays a line of the following format:

**Simulation Cover : n %**

where **n** indicates the percentage of nodes in the design that have had a transition from low to high or high to low during simulation. This feature informs you of how well the design has been exercised during a particular simulation run.

Examples:

**PLOT CLOCK A? OUTPUT1 @PLOT.OUT ;**  
(creates the file **PLOT.OUT**, containing output waveforms of **OUTPUT1**, **CLOCK**, and all nodes beginning with **A** )

**PLOT CLOCK INPUT OUTPUT ;**  
(creates **<filename>.WAV**, containing output waveforms of **CLOCK**, **INPUT**, and **OUTPUT**)

**PLOT ON ;** (turns plotting on)

**PLOT OFF ;** (turns plotting off)

# QUIT

Format: 'QUIT' ;

Purpose: Terminates the current simulation session.

Description: When **QUIT** is used in interactive mode, the Functional Simulator prompts you to indicate whether you wish to terminate the simulation. If you do, you are returned to DOS or the APLUS Menu. If the Functional Simulator encounters a **QUIT** command in a Command File called up by the **EXEC** command, it closes the file and returns to take simulation command input from the file containing the **EXECUTE** command or from the keyboard.

If the Functional Simulator is running in batch mode, the **QUIT** command terminates the simulation immediately.

Example: **QUIT** ;

## RESTORE (REST)

Format:           **'REST' [@<pathname>] ;**

Purpose:           Restores the state of the Functional Simulator that was saved in the Save File when the **SAVE** command was last issued. The default filename is **<JEDEC filename>.SAV**. This command is optional.

Description:     **RESTORE** loads the Save File named by **@<pathname>** and you may resume simulation from the state that was saved, or change parameters and begin/continue simulation.

Example:          **REST @sim.sav ;**  
                    (restores simulation environment and state  
                    from the file **sim.sav**)



# SAVE

Format: **'SAVE' [@<pathname>] ;**

Purpose: Creates a file that saves the simulation environment and state so that it may be restored later with the **RESTORE** command. The default filename is **<JEDEC filename>.SAV**. This command is optional.

Description: Saves the current simulation environment in the specified Save File. **SAVE** does not affect the simulation process and simulation may continue from this point.

Example: **SAVE @sim.sav ;**  
(saves simulation environment and state to the file **sim.sav**)

## **SIMULATE (SIM)**

Format:           **'SIM'** [**<cycle value>**] ;

Purpose:           Runs the Functional Simulator. The specified cycle is the cycle at which the Functional Simulator will stop. This command is required.

Description:      If a cycle value is not specified, the **SIM** command executes for one cycle. In conjunction with a cycle value, **SIM** executes to the specified cycle.

You may interrupt simulation by pressing **<Esc>**.

Examples:        **SIM** ;  
                  **SIM 10** ;   (Simulator executes to cycle **10**)  
                  **SIM +10** ; (Starting at the current cycle, the Simulator executes **10** cycles )

# STATUS

Format: `'STATUS' ;`

Purpose: Displays information on the state of the simulation. This command is optional.

Description: **STATUS** informs you of the JEDEC filename and whether the **PLOT**, **WATCH**, and **LOG** commands are turned on or off.

Like the Watch and Plot files, **STATUS** displays a line of the following format:

**Simulation Cover : n %**

where **n** indicates the percentage of nodes in the design that have had a transition from low to high or high to low during simulation. This feature informs you of how well the design has been exercised during a particular simulation run.

Example: `STATUS ;`

## SYMBOLS (SYMB)

Format:            **'SYMB' <node list> ;**

Purpose:            Displays a list of node names used in the design.

Description:      This command may be used to obtain a list of all predefined node names for each element of BUSTER (EPB1400) buses, pins and macrocells, as well as of user-assigned node names. Buses are given predefined names **.B0** to **.B7**; macrocells are given predefined names **.M1** to **.M#**, where # is a 1- or 2-digit number. Pins are given predefined names **.P1** to **.P#**, where # is a 1- or 2-digit number. (EP1800G pin names also include a letter, e.g., **.PA10**.) Predefined node names may be used to reference buried registers and other unnamed nodes.

Pattern matching with the wildcard characters **?** and **\*** may be used with the **SYMB** command.

Examples:        **SYMB \*;**            (displays all predefined and user-assigned node and group names)  
**SYMB T\* ;**        (displays all user-assigned node and group names starting with T)  
**SYMB .M\* ;**        (displays all predefined macrocell names)

## VECTOR (VEC)

- Format: 'VEC' [@<pathname>] ;
- Purpose: Tells the Functional Simulator which Vector File to use for input vectors. The default filename is <JEDEC filename>.VEC. If an extension is not specified, the filename defaults to .VEC. This command is required.
- Description: The Functional Simulator uses vectors that define the input levels for the design to be simulated. Therefore, if there is no PATTERN command, VEC must be used at least once before starting the simulation. The VEC command may be used repeatedly, with each new command indicating a change in the source of vector definitions (i.e., a different Vector File). Each time the Functional Simulator encounters the VEC command, it closes the current Vector File (if one exists) and opens the requested file. It builds a table of vector levels based on the contents of the file and refers to this table whenever a vector is required for simulation.
- Example: VEC @sim.VEC ;  
(read input vectors from the file sim.VEC)
- Note: Refer also to the detailed description under *Vector File Input*.

# VIEW

Format: **'VIEW'** ;

Purpose: Invokes the Virtual Logic Analyzer.

Description: Allows you to view simulated output waveforms interactively in a graphical display. The **PLOT** command specifies which signals are to be monitored by the Virtual Logic Analyzer.

Extensive on-line **HELP** files describe every subcommand available with **VIEW**.

Examples: **VIEW** ;



Refer to the section tabbed *Virtual Logic Analyzer* for a detailed description of this command.

# WATCH

Format: **'WATCH' 'ON' | 'OFF' | <node list> [@<pathname>] ;**

Purpose: Prints the logic levels of the nodes in table form (in numbers, X's, and Z's). The default filename is **<JEDEC filename>.TBL**. This command is optional.

Description: Node levels are printed in the order of the node list and according to base specifications (i.e., binary, octal, decimal, or hexadecimal values). If a filename is included, output is sent to the specified file; otherwise, it goes to the default **<JEDEC filename>.TBL**. Logic levels of nodes printed are those reached after the design has stabilized following the clocking of the input vector.

Like **PLOT** and **STATUS**, **WATCH** displays a line of the following format:

**Simulation Cover : n %**

where **n** indicates the percentage of nodes in the design that have had a transition from low to high or high to low during simulation. This feature informs you of how well the design has been exercised during a particular simulation run.

Examples: **WATCH CLOCK A? OUTPUT1 @watch.out ;**  
(creates the file **watch.out**, containing tabular output of **OUTPUT1**, **CLOCK**, and all nodes beginning with **A** )  
**WATCH CLOCK INPUT OUTPUT ;**  
**WATCH ON ;** (turns watching on)  
**WATCH OFF ;** (turns watching off)

Note: Pattern matching with the wildcard characters **?** and **\*** may be used with **WATCH**. For example, if you enter **WATCH \*E**; all nodes with names up to eight characters long and ending with **E** are displayed.





# Virtual Logic Analyzer (VIEW Command)

---

The Virtual Logic Analyzer (VLA) enables you to interactively view and analyze the node waveforms of your design. It is invoked with the **VIEW** command and incorporates a variety of features. The following pages give a detailed description of VLA features and a list of all available commands.

# Functional Description

The Virtual Logic Analyzer uses a data buffer that contains all node level and timing information generated by the Functional Simulator (This information is also included in the Plot File.) This information may be displayed in three independent windows, as illustrated in Figure LA--1.

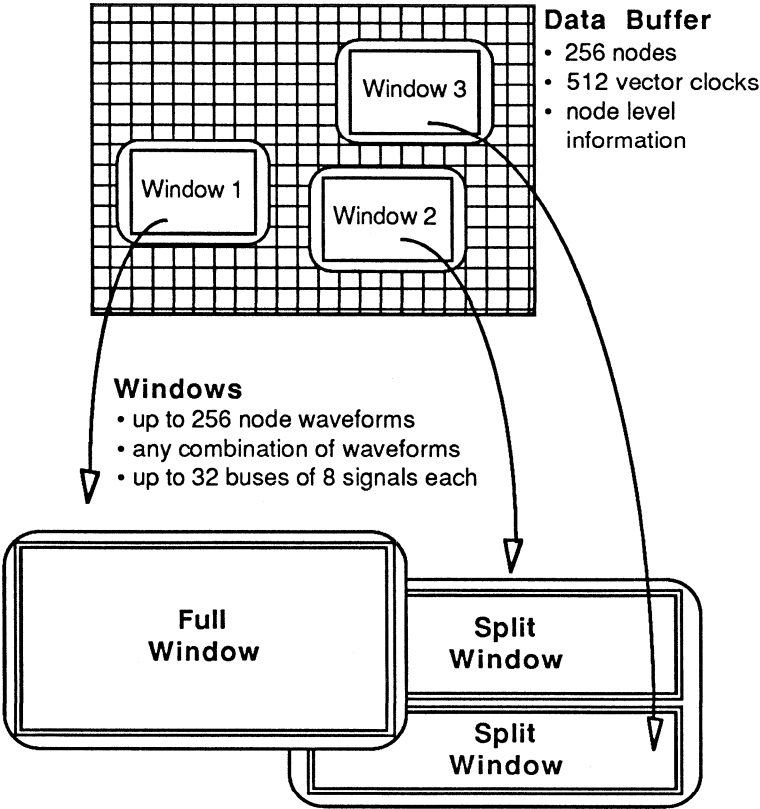


Figure LA-1. Virtual Logic Analyzer

Each window is completely independent and may display any combination of information contained in the data buffer.

The data buffer may contain up to 512 vector clocks; for example, if you simulate to 1000 vector clocks, only the most recent 512 vector clocks will be stored in the data buffer.

The **PLOT ON** and **PLOT OFF** commands determine the range of the simulation that should be output in waveform, i.e., they control the number of vector clocks for which the specified node level information is stored in the data buffer. Using **PLOT ON** and **PLOT OFF** more than once in one file creates discontinuities in the cycle value of adjacent waveform segments. Such discontinuities are marked with a flashing character in the waveform. As you move the waveform cursor across this flashing character, the increment of the cycle number located on the dotted line indicates the the discontinuity in the waveform. The character flashing may be toggled **OFF** by pressing **<f>**.

The Virtual Logic Analyzer provides two main facilities, each of which is accompanied by extensive pop-up help windows:

- (1) Manipulation of the window display by
  - zooming
  - panning
  - searching for a specified set of node levels
  - splitting the screen horizontally into two windows
- (2) Manipulation of the displayed node waveforms by
  - reordering nodes
  - combining nodes into a bus or expanding the bus into its individual nodes
  - adding and deleting nodes from the display
  - highlighting breakpoints

# Invoking the Virtual Logic Analyzer

The Virtual Logic Analyzer is set up with the **PLOT** command. After entering the **PLOT** command to specify the nodes you wish to view, followed by the **SIM** command, you type at the prompt:

```
0> VIEW <Enter>
```

The main **VIEW** screen is displayed, as shown in Figure LA-2.



You may use the **VIEW** command when the Functional Simulator is run in batch mode. Simply type the command into the input file (.CMD) along with the other commands you wish to enter.

# Full Window

The full window screen, shown in Figure LA-2, is divided into display fields that show the current status of the Virtual Logic Analyzer and window configuration.

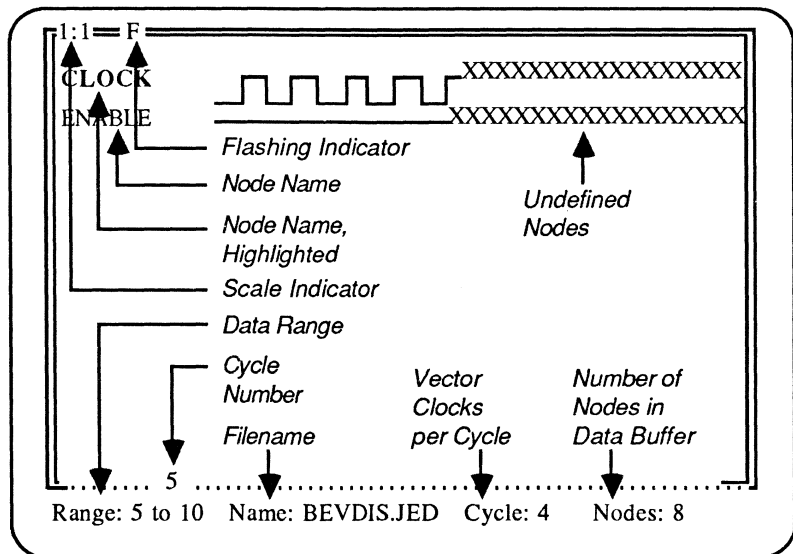


Figure LA-2. Full Window Screen

The fields are as follows:

**Scale Indicator** Shows the scale of waveform characters per vector clock. Four scales are available: 1:1, 1:2, 1:4, and 1:8. At the scale of 1:1 (the default), one waveform character equals one vector clock. At the 1:8 scale, one character equals eight vector clocks and the entire data buffer (512 vector clocks) is displayed on the screen.

**Flashing Indicator** Indicates whether flashing is ON (the default) or OFF. Flashing occurs at the vector clocks where a breakpoint has been hit or where the waveform has been discontinued and then restarted at higher vector clocks (i.e., in the case of two adjacent but not contiguous vector clocks).

**Node Name** Any one of the nodes specified in the PLOT command and selected for display in the waveform list. The waveform list may contain any combination of nodes and buses in any order up to 256 waveforms. Each node may appear any number of times. Up to 11 nodes may be displayed on the screen at any one time.

**Node Name, Highlighted** Indicates that a node has been marked with the <↑> or <↓> key. When you press <Space>, the node is selected and may now be moved, duplicated, deleted, or grouped into a bus.

**Data Range** Indicates the range of valid vector clocks that have been captured in waveform. Each node may have information representing up to 512 vector clocks. Node level transitions at larger scales (e.g., 1:8) that are lost due to poor display scaling are shown with parallel vertical lines (||) in the waveform.

**Cycle Number** Indicates the vector clock cycle where the cursor is currently located.

**Filename** Indicates the design JEDEC filename.

**Vector Clocks per Cycle** Indicates how many vector clocks have been specified per cycle length. For example, "Cycle: 4" indicates that the cycle number is incremented at every fourth vector clock.

**Number of Nodes in Data Buffer** Indicates the total number of nodes in the data buffer that contains all node level and timing information. Up to 256 nodes with 512 vector clocks each may be in the buffer and may be called up for display in any one of the three windows.

# Window Display Control

---

The Virtual Logic Analyzer enables you to manipulate the window layout and configuration so that you may quickly display and easily move around the contents of the entire data buffer. The major windowing features are zooming, panning, searching for sets of node logic levels, and splitting the window.

## Zooming

You may zoom to any one of four display scales with the <PgUp> and <PgDn> keys. The scale indicator in the upper-left-hand corner of the window display shows the current scale.

- At the 1:1 scale (the default), one character represents one vector clock; 64 vector clocks are displayed at any one time.
- At the 1:2 scale, one character represents two vector clocks; 128 vector clocks are displayed at any one time.
- At the 1:4 scale, one character represents four vector clocks; 256 vector clocks are displayed at any one time.
- At the 1:8 scale, one character represents eight vector clocks, and 512 vector clocks—i.e., the entire data buffer—are displayed.



Vector clocks are compacted as you zoom out. A node level that changes during the period represented by one character at a particular zooming scale (i.e., within 2, 4, or 8 vector clocks if the scale is 1:2, 1:4, or 1:8, respectively) will disappear from the screen. Such a node transition is shown in the waveform with pairs of vertical lines (||).

# Panning

The Virtual Logic Analyzer allows you to pan across the entire data buffer.

Three cursors, shown in Figure LA-3, are provided for panning:

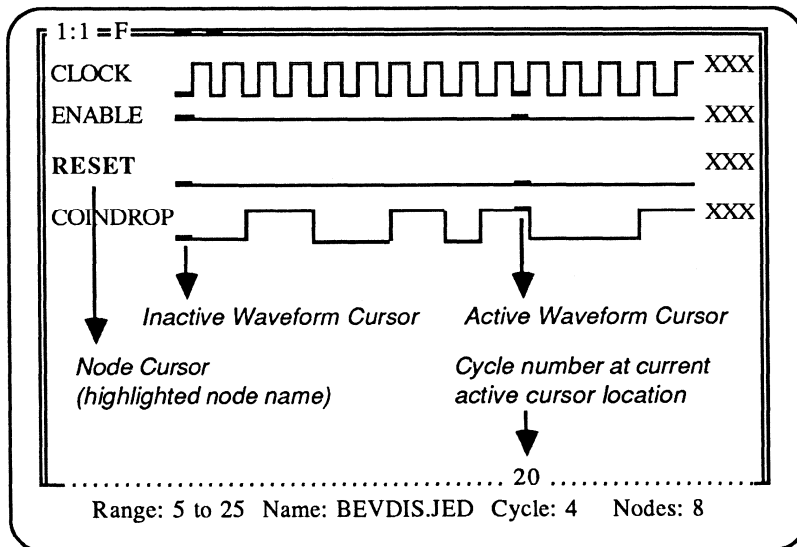


Figure LA-3. Node and Waveform Cursors

## Node Cursor

<↑> or <↓> moves the node cursor vertically from node name to node name. The current cursor location is displayed as a highlighted node name.



After you highlight a node name with the node cursor and press **<Space>**, the node name begins flashing. The flashing indicates that the node has been selected and will remain so even if you move the node cursor. This process allows you to target and select one or more nodes for reordering, deleting, or grouping into a bus. Refer to Table LA-1 for a summary of the commands that move the node cursor.

## Waveform Cursors

Two horizontal waveform cursors are provided: one active and the other inactive. On a black/white display screen, the location of these cursors is indicated by highlighted characters in the waveforms. On a color screen, the two cursors have different colors.

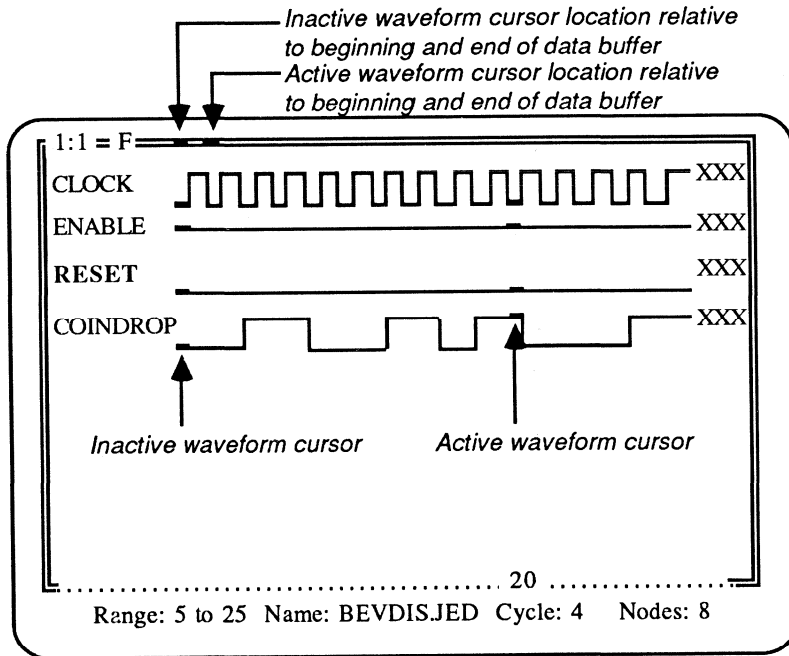
The inactive cursor is locked in place, while the active cursor may be moved left or right. (Refer to Table LA-1 for a summary of the commands that move the waveform cursor.)

When you press **<\*>**, the roles of the cursors are reversed so that the active cursor becomes inactive and vice versa. This feature allows you to jump back and forth between two sections of the data buffer.

The cycle of the vector clock associated with the active waveform cursor is always displayed on the dotted line at the bottom of the window and moves with the cursor. The timing associated with this cycle may be measured relative to time 0 (an absolute number) or relative to the inactive cursor (a relative number).

**Example:** If the inactive cursor is located at cycle 10 and the active cursor is at cycle 20, the number 20 is displayed at the bottom of the window, indicating that you are at cycle 20 of the waveform. If you press **<Enter>**, this number changes to +10, indicating that the active cursor is 10 cycles beyond the inactive cursor. If you press **<\*>**, the active cursor becomes inactive, and the new active cursor is located at cycle -10, namely 10 cycles before the inactive cursor. This feature enables you to quickly measure the length of node pulses.

Highlighted dashes on the top window border indicate where these two horizontal cursors are located relative to the beginning and end of the entire data buffer. Refer to Figure LA-4.



**Figure LA-4. Relative Cursor Location**

## Searching

The Virtual Logic Analyzer also provides a Search command (<s>) to search for a specified set of node levels. When <s> is invoked, you are prompted for a list of node levels. The first node you enter is the one highlighted with the node cursor; the next one is the one immediately below, etc.

To locate all instances of a particular set of node levels in the entire data buffer, use the Repeat Search command (<r>).

## Splitting the Window

The split window facility lets you split the display screen horizontally.

To split the window, press <Del> while the main window is displayed. The window is split as shown in Figure LA-5.



Because the windows are independent of each other, you may:

- Zoom to the smallest scale in one window and to the largest in the other.
- Change the size of a window to display additional waveforms, up to a maximum of 10 displayed in each split window.
- Use the node and waveform cursors in each window, or lock them (<L>) so they move concurrently in each window.
- Display a total of 32 buses, combining up to 8 node waveforms each, in any one window or combination of windows at any one time

# Node and Bus Control

---

Nodes and their associated waveforms may be tagged and selected for quick and easy manipulation such as adding, deleting, reordering, and combining into a bus.

## Nodes

Each window contains a waveform list which may consist of any combination of the nodes specified in the data buffer (maximum 256 nodes). A node may appear any number of times as long as the entire list does not exceed the limit of 256 nodes.

**Move** a node by first marking it with the node cursor (<↑> or <↓>). Then press <Space> to actually select the node (it will continue to flash even when you move the node cursor to another node). After the node is selected, you may move it with the Move command (<m>); it will then be placed above the highlighted node.

*Example:* To move the node **COINDROP**, shown in Figure LA-3, so that it immediately follows **CLOCK**, you highlight **COINDROP**, press <Space>, move the node cursor to **ENABLE**, and then press <m>.

**Delete** a node by first selecting it with <Space>. Then press <d> (the Delete command). (Note: Press <u> to undo any command that changes the order or grouping of nodes.)

**Add** a node from the data buffer with the Add command (<a>). Pattern matching with the wildcard characters \* and ? may be used to add more than one node.

*Example:* To add all nodes in the data buffer that start with "D" to the window, you highlight the node before which the new nodes should be inserted, press <a>, and type **D\*** <Enter> into the prompt box:

Refer to Table LA-1 for a summary of available commands.

## Buses

The Bus command (<b>) enables you to create buses with any combination of the nodes stored in the data buffer. Up to eight nodes may be combined into a bus. Refer to Figure LA-6, which shows one window with individual nodes and another with four nodes combined into a bus. The most significant bit is always at the top of the bus; the least significant is at the bottom.

Buses may be merged with other nodes and buses provided the total number of bits is no greater than eight. If bus waveforms are placed next to one another, their values may be combined and read as one. With this “vertical stacking” feature, you can in effect create buses that are wider than eight bits. You may create up to 32 buses in any one window or combination of windows at any one time.

The value of a bus is displayed in hex numbers. It is displayed only when the value changes, thus indicating the transition points of the bus. A bus value that changes during the period represented by one character at a particular zooming scale (i.e., within 2, 4, or 8 vector clocks if the scale is 1:2, 1:4, or 1:8, respectively) will disappear from the screen. This compaction of values is shown in the waveform with pairs of vertical lines (||).

High-impedance buses are shown with two vertically stacked ‘z’s and unknown bus values are shown with two vertically stacked ‘x’s.

The Expand Command (<e>) allows you to expand a bus into its individual nodes. After expanding the bus, you may undo this command by pressing <u> to redisplay the bus. This technique allows you to quickly view the nodes that comprise a particular bus.

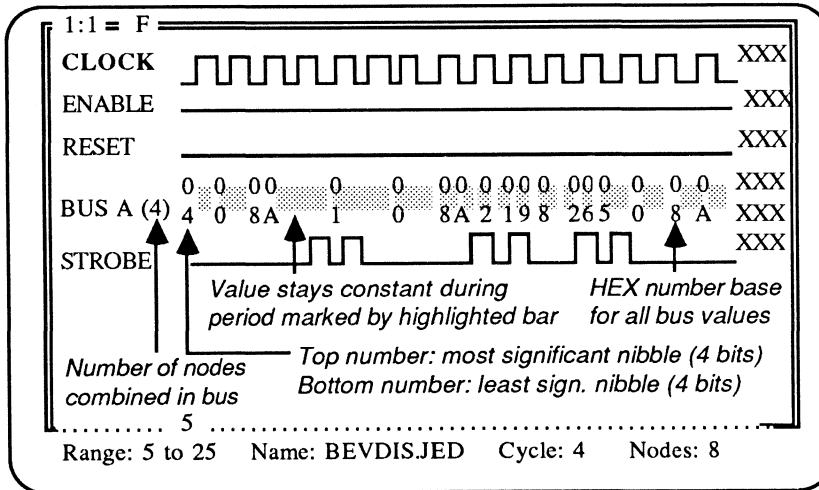
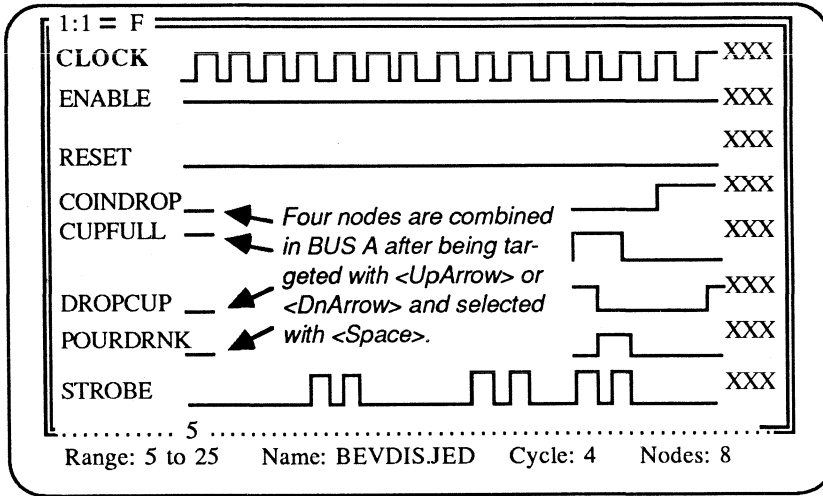


Figure LA-6. Bus Waveform



# Summary of VIEW Subcommands

The **VIEW** command provides a variety of subcommands. These subcommands fall into two categories: those that manipulate the window display and those that manipulate the node waveforms. Table LA-1 summarizes all subcommands; it is followed by a detailed explanation of the commands in each category.

**Table LA-1. VIEW Subcommands (Part 1 of 2)**

| WINDOW MANIPULATION |              |                                             |
|---------------------|--------------|---------------------------------------------|
| CATEGORY            | KEY(S)       | FUNCTION                                    |
| Cursor Moves        | <←>          | Move left 1 character                       |
|                     | <→>          | Move right 1 character                      |
|                     | <Home>       | Move to left window edge                    |
|                     | <End>        | Move to right window edge                   |
|                     | <Ctrl><←>    | Jump left 8 characters                      |
|                     | <Ctrl><→>    | Jump right 8 characters                     |
|                     | <Ctrl><Home> | Move to oldest vector clock                 |
|                     | <Ctrl><End>  | Move to newest vector clock                 |
|                     | <+>          | Center display on cursor                    |
|                     | <*>          | Switch cursors                              |
|                     | <c>          | Toggle black&white/color window and display |
|                     | <l>          | Lock cursors in split window                |
|                     | <p>          | Print the window                            |
|                     | <r>          | Repeat search for node levels               |
|                     | <s>          | Search for node levels                      |
|                     | <v>          | Vertically compress the window              |
| Zooming             | <PgUp>       | Zoom out                                    |
|                     | <PgDn>       | Zoom in                                     |
| Window Split        | <Shift><↑>   | Move split window boundary up               |
|                     | <Shift><↓>   | Move split window boundary down             |
|                     | <Del>        | Toggle window split                         |
|                     | <Ins>        | Select active window                        |

**Table LA-1. VIEW Subcommands (Part 2 of 2)**

| <b>WAVEFORM MANIPULATION</b>            |               |                                            |
|-----------------------------------------|---------------|--------------------------------------------|
| <b>CATEGORY</b>                         | <b>KEY(S)</b> | <b>FUNCTION</b>                            |
| <b>Nodes</b>                            | <↑>           | Move node cursor up 1 node                 |
|                                         | <↓>           | Move node cursor down 1 node               |
|                                         | <Ctrl><PgUp>  | Scroll up one display window               |
|                                         | <Ctrl><PgDn>  | Scroll down one display window             |
|                                         | <Space>       | Select node (flashing)                     |
|                                         | <Enter>       | Toggle relative and absolute cursor time   |
|                                         | <a>           | Add node(s)                                |
| <b>Buses</b>                            | <d>           | Delete node(s)                             |
|                                         | <f>           | Toggle flash for highlighted vector clocks |
|                                         | <m>           | Move 1 or more selected nodes              |
|                                         | <b>           | Create a bus                               |
|                                         | <c>           | Expand a bus                               |
| <b>Undo<br/>Quit Help<br/>Quit VIEW</b> | <u>           | Undo previous command                      |
|                                         | <Esc>         | Exit Help file                             |
|                                         | <q>           | Exit Virtual Logic Analyzer                |



If you misspell a command, a summary of VLA commands is automatically displayed. This Help window provides access to on-line information about each VLA command.

## Window Manipulation Commands

The windowing facility lets you examine any portion of the simulated design, regardless of the screen limitations. These commands make it easy to move around the display window.

Window manipulation commands enable you to:

- move the cursor
- zoom in and out
- vertically compress the screen display
- split the window
- print the window

Each command is described in detail below. Refer to Table LA-1 for a command summary.

**<←>/<→>** Moves the active waveform cursor to the left/right by one character and scrolls the window if necessary. Depending on the current scale setting, this command may move the cursor up to eight vector clocks.

**<Home>/<End>** Moves the active waveform cursor to the leftmost/rightmost position of the node waveforms displayed on the screen.

**<Ctrl><←>/<Ctrl><→>** Moves the active waveform cursor to the left/right by eight characters and scrolls the window if necessary. Depending on the current scale setting, this command may move the cursor up to 64 vector clocks.

**<Ctrl><Home>/<Ctrl><End>** Moves the active waveform cursor to the oldest (first)/newest (last) vector clock in the waveform buffer.

**<+>** Centers the window display on the active waveform cursor.

**<\*>** Toggles the state of the waveform cursors: the inactive waveform cursor becomes active and vice versa.

- <c>** Toggles between black/white and color display. If your terminal simulates color with gray scales, we recommend that you turn color simulation **OFF**.
- <l>** In split window mode, **<l>** locks the two active cursors together so that when you move the cursor in the active window, it moves proportionately in the inactive one. This feature allows you to simultaneously scroll through the same set of nodes at two different display scales. An **L** displayed on the left-hand side of the upper window indicates that the cursors are locked.
- <p>** Prints the window directly to your printer or saves a screen dump to a file. After you press **<p>**, you may type a pathname or simply press **<Enter>** to print the screen.
- <r>** Repeats the most recent search of node levels. (Refer to the description of **<s>** for details.) This command enables you to quickly locate all occurrences of a particular node set.
- <s>** Searches for a set of specified node levels. When **<s>** is invoked, you are prompted for a list of node levels. The first node level you enter must be the one highlighted by the node cursor; the second must be the one immediately below, etc. Signals may not be skipped, so you may need to reorder nodes before entering the list.
- <v>** If you have an Enhanced Graphics Adapter (EGA) card, this command toggles the screen display between 25 and 43 lines. This command allows you to display additional waveforms on the screen.
- <PgUp>/<PgDn>** Zooms to any one of four display scales. The scale indicator at the upper-left-hand corner of the screen displays the current scale.
- At the 1:1 scale (the default), one character represents one vector clock; 64 vector clocks are displayed at any one time.
  - At the 1:2 scale, one character represents two vector clocks; 128 vector clocks are displayed at any one time.

- At the 1:4 scale, one character represents four vector clocks; 256 vector clocks are displayed at any one time.
- At the 1:8 scale, one character represents eight vector clocks, and 512 vector clocks—i.e., the entire data buffer—are displayed.

**<Shift><↑>/<Shift><↓>** Moves the dividing boundary of the split window up or down one node so that the sizes of the two windows may be adjusted. In split window mode, a window may display a maximum of 10 nodes.

**<Del>** Toggles between split window and full window mode.

**<Ins>** Toggles the split windows from active to inactive.

## Waveform Manipulation Commands

The nodes stored in the data buffer and their associated waveforms may be grouped, merged into buses, and deleted from or added to the window display.

Waveform manipulation commands enable you to:

- manipulate nodes
- manipulate buses

Each command is described in detail below. Refer to Table LA-1 for a command summary.

**<↑>/<↓>** Moves the node cursor up/down one node.

**<Ctrl><PgUp>/<Ctrl><PgDn>** Moves the node cursor up/down by one displayed screen (i.e., one window's worth) of information.

- <Enter>** Toggles the absolute and relative timing information associated with the active cursor. The absolute value is that which is measured from vector clock 0 to the vector clock at the current active cursor location. The relative value is the difference between the value at the inactive cursor location and the value at the current active cursor location. When **<Enter>** is used with the Switch Cursors command (**<\*>**), you may quickly measure the timing difference between two sets of vector clocks.
- <Space>** Selects a marked node. After a node is marked with the node cursor (i.e., highlighted) and you press **<Space>**, the node flashes. The flashing indicates that the node has been selected and may now be moved, deleted, or merged into a bus. A node is “unselected” (i.e., the flashing stops) if you press **<Space>** again.
- <a>** Adds a selected node from the data buffer to the window display. First place the node cursor on the node which will immediately follow the node(s) to be inserted. Then press **<a>** and type one or more node names into the prompt box. Press **<Enter>**. You may enter up to 256 nodes or 32 buses comprised of these nodes; nodes and buses may appear more than once. Signal names specified may include the wildcard characters **\*** and **?**.
- <d>** Deletes one or more nodes or buses from the window display. First select the node(s) and/or bus(es) you wish to delete with the node cursor, then press **<d>**.
- <f>** Toggles flashing of node characters (default = **ON**). The **F** on the left-hand side of the top window border indicates that the flashing feature is turned on. Breakpoints and non-contiguous vector clocks are identified with a flashing waveform character. If you press **<f>**, the flashing stops and the **F** on the window border is deleted. However, even if node flashing is turned off, a flashing segment on the window border remains, indicating the location of the breakpoints and non-contiguous vector clocks relative to the beginning and end of the data buffer.

**<m>** Moves a selected node. After a node name is marked with the node cursor (i.e., highlighted) and selected with **<Space>**, it flashes. You then move the node cursor to the node or bus name before which you wish to place the selected node and press **<m>**. You may move any number of nodes and buses, which will retain the same sequence they had in the waveform list.

Buses are selected, moved, and deleted with the same commands as nodes. Only the following two commands are bus-specific:

**<b>** Forms a bus by merging all selected nodes into one bus waveform. The nodes to be merged must first be selected with **<Space>** (i.e., so that they are flashing). Then you place the node cursor on the node before which you wish to place the bus and press **<b>**. The prompt box prompts you for a bus name. Type the name and press **<Enter>**.

The most significant nibble (4 bits) is at the top of the display, the least significant nibble is at the bottom.

Since groups cannot be plotted, this command allows you to re-form the Buster (EPB1400) internal bus from its individual nodes (.B0 through .B7).



A bus may contain 1 to 8 nodes. If you need a wider bus, simply create a second bus with additional nodes and place the buses next to each other.

**<e>** Expands a highlighted bus into its individual nodes. When used with the Undo command (**<u>**) the Expand command may be used to quickly view the members of a bus and then merge them again.

To undo a command, quit a Help file, or exit the Virtual Logic Analyzer, the following commands are available:

- <u>** Removes the effect of the most recent **<a>**, **<d>**, **<b>**, or **<e>** command. Even if you quit the Virtual Logic Analyzer and then reenter it with the **VIEW** command, the Undo command will undo the most recent one of these commands. Note also that when this command is used with the Expand command (**<e>**), it provides a quick way to expand a bus to view its individual members and then merge them together again.
- <Esc>** Exits a Help file in the Virtual Logic Analyzer.
- <q>** Quits the Virtual Logic Analyzer and returns you to the Functional Simulator.



# Functional Simulator Messages

This section alphabetically lists the error and warning messages generated by the Functional Simulator (FSIM). Each entry describes the likely cause of the message and, when appropriate, gives suggestions for corrective action. (Messages with the prefixes **ERR-**, **INFO-**, and **WARN-** are listed in *A+PLUS Messages* in the ***A+PLUS Reference Guide.***)

Functional Simulator error messages indicate errors that must be corrected before simulation can continue, while warning messages indicate potential problems which may require corrective action.

When the Functional Simulator is run in batch mode, error messages are preceded by the line number in the Command file that contains the error, in the following format:

**<filename>.CMD line # - <error message>**

All warning messages begin with **Warning:**

# Error Messages

## Bad cycle value

- CAUSE:** You have entered a cycle value that contains non-decimal numbers or is less than the current cycle value.
- ACTION:** Enter a cycle value consisting of decimal numbers greater than or equal to the current cycle value.

## Bad group value

- CAUSE:** The Functional Simulator found a group value that had too many digits or digit(s) of the wrong base.
- ACTION:** Ensure that the group value is expressed in the same base (i.e., decimal, octal, hexadecimal, or binary) as the base specified in the **GROUP** command.

## Bad JEDEC file

- CAUSE:** The specified JEDEC file has been corrupted or was not created by the Altera Design Processor.
- ACTION:** Resubmit your design file to the ADP and generate a new JEDEC file. (Note: only JEDEC files produced by the ADP may be used with the Functional Simulator.)

## Bad node value

- CAUSE:** A value for a node was specified in a node value list that contained a character other than **0**, **1**, **X**, or **Z**.
- ACTION:** Reenter the node value with one of the legal characters **0**, **1**, **X**, or **Z**.

## Bad repeat factor

- CAUSE:** The Functional Simulator found a repeat factor greater than 10,000 or which contained a non-decimal character in a vector pattern.
- ACTION:** Ensure that repeat factors are less than 10,000 and contain only decimal numbers.

### **Breakpoint number does not exist**

- CAUSE:** You tried to reference a breakpoint with a breakpoint number that does not exist.
- ACTION:** Use the **BREAK** command to show a summary of the existing breakpoints.

### **Can't continue -- specify new cycle value with SIM**

- CAUSE:** You have attempted to execute a **CONTINUE** command, but the Functional Simulator has already reached the cycle specified in the most recent **SIMULATE** command.
- ACTION:** Use the **SIMULATE** command (with an optional cycle value) to continue simulation.

### **Can't open <name> file**

- CAUSE:** After checking the specified directory and/or current DOS path, the Functional Simulator could not find the specified file (i.e., JEDEC, Vector, Command, or JEDEC Description File [JDF]) for the design. This message may also indicate that the file is corrupted; the disk is full, write-protected or corrupted; or that available memory is insufficient to open the file. (In batch mode this error is fatal.)
- ACTION:** Check the path/filename and ensure that you have 128K of disk space, enough memory to run A+PLUS, and that the disk is not corrupted or write-protected. If the file specified is a JDF, move the file to the **APLUS** directory. (You may recopy the JDF from the distribution diskette.)

### **Can't plot groups**

- CAUSE:** A group was found in a **PLOT** command node list. Groups cannot be output in waveform.
- ACTION:** Plot the individual members of the group. If the group you wished to plot was a **BUSTER** internal bus, plot the individual elements of the bus with the predefined names **.B0** to **.B7**.

### **Can't restore—JEDEC file changed or Save file corrupted**

- CAUSE:** The Save File named in a **RESTORE** command could not be used to restore the state of the Functional Simulator either because the Save File has been corrupted, or a different/changed JEDEC File is being used. Note that files created by the **SAVE** command are associated with the JEDEC file that is in use when the **SAVE** command is executed.
- ACTION:** Ensure that you have used the Save File that corresponds to the JEDEC file you wish to simulate. (If you have changed the JEDEC file since the Save File was created, you cannot restore the saved environment; you must begin a new simulation session.)

### **Can't simulate without input vectors**

- CAUSE:** A **SIM** command was executed before a **VECTOR** or **PATTERN** command. Since no input vectors have been defined, simulation cannot begin.
- ACTION:** Before using the **SIM** command, use the **VECTOR** command to specify a Vector File or enter input vectors with the **PATTERN** command.

### **Command ended prematurely**

- CAUSE:** A command terminator (either **EOL** in interactive mode or a semicolon in batch mode) was found before the command was complete.
- ACTION:** Reenter the command. Remember that commands may span lines only in batch mode, not interactive mode.

### **Command syntax error**

- CAUSE:** The command line contains a syntax error.
- ACTION:** Refer to the section entitled *Command Reference* for the correct syntax of all Functional Simulator commands.

### **Command terminator (<Enter> or ';') expected**

- CAUSE:** A command terminator (either **<Enter>** in interactive mode or a semicolon in batch mode) was not found at the end of a complete command.
- ACTION:** Terminate all commands with **<Enter>** (in interactive mode) or a semicolon (in batch mode).

### **Corrupted Save file**

- CAUSE:** You cannot use the Save File specified in a **RESTORE** command because it is either corrupted or was not produced by a Functional Simulator **SAVE** command.
- ACTION:** Enter new simulation conditions.

### **Cycle range expected**

- CAUSE:** The keyword **RANGE** in the **BREAK** command was not followed by a cycle range.
- ACTION:** Enter a cycle range after each **RANGE** keyword.

### **Equals sign expected**

- CAUSE:** An equals sign (=) was expected but not found (e.g., in a group, cycle, or node value list definition).
- ACTION:** Refer to the section entitled *Command Reference* for the correct syntax of all Functional Simulator commands.

### **Group names cannot have subnode extensions**

- CAUSE:** A subnode extension was used on a group name. You may only reference the subnodes of an individual pin.
- ACTION:** Specify the individual members of the group with subnode extensions.

### **Groups may not be nested**

- CAUSE:** A group name was found in a group member list. A group may not contain another group.
- ACTION:** Redefine groups to eliminate nesting.

### **I/O error**

- CAUSE:** A Functional Simulator input file can't be found anywhere on your DOS path, can't be opened/read/closed because available memory is insufficient or the disk is full or corrupted, or the file is corrupted.
- ACTION:** Ensure that you have 128K of disk space and enough memory to run A+PLUS, that your DOS path includes the current directory, that the diskette is not write- or read-protected, corrupted, or not in the disk drive.

### **Incompatible or obsolete <str> file**

- CAUSE:** When the Functional Simulator opened the specified file (i.e., a JEDEC Description File [.JDF] or Part Description File [.PDF]), it found that the file was corrupted or from an earlier version of A+PLUS.
- ACTION:** Reinstall the JDFs/PDFs (provided on the distribution diskettes) in the APLUS directory on your computer. If this error message persists, contact Altera Applications.

### **INIT command allowed only at cycle=0**

- CAUSE:** An INIT command was executed when the cycle value was greater than 0.
- ACTION:** Use the FORCE command instead of INIT when the cycle value is greater than 0.

### **Internal error: <str>**

- CAUSE:** There is an internal error.
- ACTION:** Should you encounter this message, please contact Altera Applications and describe the error number/text and the conditions under which it occurred.

### **Invalid commands in breakpoint command list**

- CAUSE:** The SIM, PATTERN, or QUIT command was encountered in a breakpoint command list.
- ACTION:** Remove all SIM, PATTERN, and QUIT commands from breakpoint command lists.

### **Invalid group name**

- CAUSE:** A group definition contains an invalid group name. This name may have been a node name or a Functional Simulator reserved word.
- ACTION:** Rename the group. (Refer to the section entitled *Reserved Words*.)

### **Line too long**

- CAUSE:** While reading input, the Functional Simulator encountered a line containing over 256 characters.
- ACTION:** Insert a line break to reduce the line length to 256 characters or less.

## **LOG, PLOT, WATCH, VECTOR, or @<pathname> expected**

- CAUSE:** The keyword **LOG**, **PLOT**, **VECTOR** or **WATCH**, or an **@** followed by a pathname was expected after the **DISPLAY** command.
- ACTION:** Enter one of the four keywords to display the Log, Plot, Vector or Watch file for the current design. Enter **@<pathname>** to display any ASCII file.

## **Mismatched parentheses**

- CAUSE:** A left or right parenthesis is missing from a vector pattern.
- ACTION:** Check the location of parentheses and ensure that there is a right parenthesis for each left parenthesis.

## **Missing '@' or bad character in pathname**

- CAUSE:** The pathname you have specified contains an illegal character (e.g., >) or is not preceded by the required **@** (at-symbol).
- ACTION:** Ensure that all pathnames are preceded by an **@**-symbol and contain only legal characters.

## **Nested infinite repeat factor**

- CAUSE:** An infinite pattern was found inside parentheses.
- ACTION:** Change the repeat pattern.

## **No Log file available**

- CAUSE:** You have attempted to display the Log File for the current design. Since you have not previously specified command logging (i.e., with **LOG ON**), the file does not exist.
- ACTION:** Use the **LOG ON** command to create a Log File.

## **No Plot file available**

- CAUSE:** You have attempted to display the Plot File for the current design. Since you have not previously specified plotting (i.e., with **PLOT ON**), the file does not exist.
- ACTION:** Use the **PLOT ON** command to create a Plot File.

### **No Vector file available**

- CAUSE:** You have attempted to display the Vector File for the current design, but the Functional Simulator has no record of the file because it has not been previously specified as the input Vector File with the **VECTOR** command. This error may also occur if you entered your input vectors with the **PATTERN** command.
- ACTION:** Specify the Vector File with the **VECTOR** command before entering the **SIM** or **DISP** commands; or, if you entered your input vectors with the **PATTERN** command, simply type **PATTERN** to display the current list of vector patterns.

### **No Watch file available**

- CAUSE:** You have attempted to display the Watch File for the current design. Since you have not previously specified **WATCH ON**, the file does not exist.
- ACTION:** Create a watch file with the **WATCH ON** command.

### **Node list empty**

- CAUSE:** The command you entered has an empty node list, so the command was ignored. This may indicate that pattern matching failed to yield any matches.
- ACTION:** Enter a node list or check your pattern matching string.

### **Node value list empty**

- CAUSE:** The node value list of a command was empty, so the command was ignored.
- ACTION:** Define node values that correspond to the node list.

### **ON or OFF expected**

- CAUSE:** You have entered a **LOG**, **PLOT**, or **WATCH** command, but there are no arguments after the command.
- ACTION:** Specified **ON** or **OFF**. You may also specify a pathname with the **LOG** command; with the **PLOT** and **WATCH** commands you may enter a node list and a pathname.



## Out of memory

- CAUSE:** The Functional Simulator dynamically allocates memory to contain items such as breakpoints or groups. This error indicates that available memory is insufficient to complete the current processing step. Other resident programs may be occupying memory, such as RAM-disks, print spoolers, communication packages, and keyboard enhancers. This message may also occur if you have used the **DOS Command (<F8>)** function and then *re-invoked* **A+PLUS** and **FSIM** from the temporary DOS environment (a duplicate copy is read into memory if you reinvoke **A+PLUS**).
- ACTION:** Try reducing the number of breakpoints or temporarily relocate other programs that are resident in memory. If you have used **<F8>**, quit **FSIM** and **A+PLUS** and type **EXIT** to return to the "original" **A+PLUS**.

## Pathname not valid in this context

- CAUSE:** A pathname, which is indicated by a leading **@**, was found when none was expected.
- ACTION:** Use the **@** (at-symbol) only as the first character of a pathname.

## Pattern syntax error

- CAUSE:** The Functional Simulator found a syntax error in a vector pattern.
- ACTION:** If your input vectors are located in a **Vector** and/or **Command** file, check the file for errors. If you entered input vectors interactively with the **PATTERN** command, you must reenter the vector pattern. (Refer also to *Legal Logic Level Characters* and *Legal Input Vector Format*.)

## Repeat operator ("\*") without parentheses

- CAUSE:** A vector pattern contains a repeat operator (**\***) that is not preceded by parentheses.
- ACTION:** Ensure that vector patterns to be repeated are enclosed in parentheses and followed by a repeat operator.

## Too many breakpoints

- CAUSE:** You have set more than 32 breakpoints, which is the maximum number allowed at any one time.
- ACTION:** To set more breakpoints, first clear some others with the **CLEAR** command.

### **Too many groups**

- CAUSE:** You have defined more than 32 groups, which is the maximum allowed at any one time.
- ACTION:** Try redefining an existing group (i.e., use an existing group name).

### **Too many members in group**

- CAUSE:** You have defined a group containing more than 32 members, which is the maximum allowed for a single group. The defined group will contain only the first 32 members specified; the remainder will be discarded.
- ACTION:** Define additional groups of up to 32 members.

### **Too many nodes in Vector file**

- CAUSE:** There are over 256 nodes in the input table of the Vector File. The Functional Simulator cannot store all the data.
- ACTION:** Use 256 nodes or less in each Vector File.

### **Too many vectors in pattern**

- CAUSE:** After repeat factors were applied, the length of a vector pattern exceeded 10,000 vectors.
- ACTION:** Reduce the pattern length to less than 10,000 vectors. You may wish to split the Vector File into two or more files and use each one in succession.

### **Unknown node name**

- CAUSE:** The Functional Simulator encountered a node name in a node list or node value list that it did not recognize.
- ACTION:** Use the **SYMBOLS** command to check whether the node exists, or make sure that you have entered appropriate pattern matching characters.

### **Unknown or invalid subnode extension**

- CAUSE:** A subnode extension was applied to a node that was either unknown or not applicable to that node (e.g., **INPUT.CLK**, where **INPUT** is a dedicated input pin.)
- ACTION:** See *Referencing Subnodes* for a description of the notation used for referencing subnodes.

### **Use only one command per line**

- CAUSE: One or more characters were found on the command line after the command terminator (;) during interactive mode simulation.
- ACTION: Enter only one command per line while running the Functional Simulator in interactive mode.

### **Vector file must begin with "PATTERN:" or "TABLE:"**

- CAUSE: The first word in a Vector File must identify the input vector format with a keyword.
- ACTION: Ensure that the **PATTERN:** or **TABLE:** keyword is the first word in a Vector File.

### **Vector table syntax error**

- CAUSE: A vector table in a Vector File contains a syntax error.
- ACTION: See *Legal Input Vector Format* for a description of vector table syntax.

## Warning Messages

**Warning: Cycle <#n>, pass <#m>, <node name>.FBK unstable. Continue?**

CAUSE: At cycle *n*, after *m* passes through the design, the level on the specified node did not stabilize. If you are in batch mode, the Functional Simulator terminates.

ACTION: If you are in interactive mode, you may type **Y** to see if the design will stabilize eventually.

**Warning: Cycle #, <node name>.CLK – undefined clock transition.**

CAUSE: The clock input for a register had an undefined transition. In the .RPT file, A+PLUS sometimes indicates that clock pins need to be connected externally. The Functional Simulator will display this message if you did not explicitly define clock vectors for the other pins.

ACTION: Ensure that the clock input is defined. For example, if the .RPT file contains a message telling you to externally connect clock pins 1 and 13, you must define vectors for both pins as follows (assuming a clock vector of **001100110011 ...**):

```
.P1 = (0011) * ;
.P13 = (0011) * ;
```

**Warning: Duplicate node name <node name> ignored**

CAUSE: The Functional Simulator has found the same node name in two places. This message may occur if <node name> was truncated to eight characters, and the truncated name is now identical to another node name.

ACTION: Use node names that contain eight characters or less.

**Warning: JEDEC symbol <node name> redefined. Use <predefined name> instead.**

CAUSE: A <node name> in the JEDEC file was the same as a predefined node name in the Functional Simulator.

ACTION: To reach the node referenced by this symbol, use the specified predefined node name.

**Warning: Node name <node name> in JEDEC file ignored – reserved word**

CAUSE: The specified <node name> in the JEDEC file is a Functional Simulator Reserved Word. The symbol was ignored.

ACTION: Use the Utilization Report (.RPT file) to find the pin or macrocell of the <node name>. To reach the node referenced by this symbol, use the predefined node name corresponding to the pin or macrocell.

**Warning: Node name <node name1> truncated to <node name2>**

CAUSE: The specified <node name1> in the JEDEC file was truncated to <node name2>, i.e., to contain eight characters, which is the maximum length permitted.

ACTION: Use node names that contain eight characters or less. Alternatively, you may use the truncated node name during simulation.

**Warning: Vector # applied to <node name> when bus transceiver state was undefined.**

CAUSE: During vector clock #, an input vector was applied to a bus port pin when the bus transceiver was in an undefined state.

ACTION: Check the Vector file and ensure that you have specified the correct vectors for all inputs that control the bus. For additional information, refer to *Bus Port Pins* in *FSIM Reference*.

**Warning: Vector # applied to <node name> when bus transceiver was driving out.**

CAUSE: During vector clock #, an input vector was applied to a bus port pin when the pin was driving out.

ACTION: Check the Vector file and ensure that you have specified the correct vectors for all inputs that control the bus. For additional information, refer to *Bus Port Pins* in *FSIM Reference*.

**Warning: Vector # applied to <node name> when output enabled.**

CAUSE: During vector clock #, input vectors were applied to bidirectional pins when the pin was not tri-stated.

ACTION: No action is required. For additional information, refer to *Bidirectional Pins* in *FSIM Reference*.

**Warning: Vector # applied to <node name> when output enable undefined.**

CAUSE: During vector clock #, an input vector was applied to a bidirectional pin when the Output Enable input of the tri-stated buffer was undefined.

ACTION: No action is required. For additional information, refer to *Bidirectional Pins* in *FSIM Reference*.

# FSIM Glossary

---

Before using the Functional Simulator, you should become familiar with the following concepts:

**breakpoint** A breakpoint is a user-defined set of conditions that allow simulation to be interrupted when the conditions are met. The breakpoint allows you to execute a list of Functional Simulator commands when simulation is halted, e.g., to display useful information at specified times during a simulation session.

**convergence** Designs containing combinatorial feedback logic may require the Functional Simulator to make several passes through the nodes in the array before a stable state can be reached, because the output of one or more primitives may depend on the feedback of another primitive (or primitives). The Functional Simulator does not assume a delay for the propagation of signals; therefore, the logic levels of feedback nodes are determined by the current input nodes, and the changes caused by the feedback nodes must also be allowed to propagate.

Convergence depends on whether or not these cycles of propagation stop, and the logic levels stabilize. Input cycles are considered to converge if no change occurs in the nodes of the array for two consecutive cycles through the array.

**Convergence Limit** The Functional Simulator limits the number of cycles through combinatorial logic that are allowed for the outputs to find a stabilizing point. In batch mode, the simulation aborts if a stable state is not reached after this number of passes; in interactive mode, you are asked whether you wish to continue.

**Command File (.CMD)** A file containing commands that guide the simulation process. It consists of a list of instructions entered with a standard text editor, and is used only when the Functional Simulator is run in batch mode. A Log File (.LOG) may be renamed as a Command File to repeat an earlier simulation run.

**cycle** A cycle is a user-defined time reference consisting of an integral number of vector clocks. The **CYCLE** command specifies the number of vector clocks in a cycle. (Refer to *Vector Processing and CYCLE Command*.)

**Functional Simulator (FSIM)** Tests the logical operation of your A+PLUS design. FSIM uses specified design and part information to model the operation of an Altera EPLD before the design is actually committed to hardware. FSIM can be run in either interactive or batch mode. It outputs a graphical waveform description of the simulated design (.WAV file), a vector table output file (.TBL), and a log of FSIM commands used for execution (.LOG file).

**group** A number of nodes within a design may be grouped so that they can be referenced, processed, and displayed as a single unit. Any command issued to a node group (e.g., **VECTOR**) will then act on all members of the group. A group may also be assigned a binary, decimal, hexadecimal, or octal number base for input and output format. (See the **GROUP** command description.)

**high-Z (tri-state)** Shown in Plot File (.WAV) and Virtual Logic Analyzer output as a Z, it indicates a high-impedance output signal from a bidirectional pin.

**Log File (.LOG)** Records the commands executed during a simulation session. This file may be renamed as a Command File (with the extension .CMD) and used in future simulation runs with the **EXECUTE** command, to duplicate all or part of the current simulation environment. Command logging may be started and stopped at any time during simulation.

**logic level** Used to define input and display output of the design being simulated. Logic levels and their corresponding symbols legal in the Functional Simulator are as follows:

|                 |                                                          |
|-----------------|----------------------------------------------------------|
| <b>1</b>        | logic high                                               |
| <b>0</b>        | logic low                                                |
| <b>0–9, A–F</b> | binary, octal, hexadecimal, or decimal values for groups |
| <b>X</b>        | undefined                                                |
| <b>Z</b>        | high impedance (tri-state)                               |



**node** All wires in a circuit that carry the same signal between components of a design are called nodes. In other words, nodes are locations within the EPLD that may attain logical levels. (Examples of nodes are the Clear (C) input on a RORF primitive or the feedback (Fbk) from a COIF primitive.) During simulation, you can symbolically reference the nodes that are connected to input pins; macrocell outputs or Fbk; buses; and D, T, Preset (P), C, Clock (Clk), Input Vector (INP)), and Output Enable (Oe) inputs to I/O primitives. Buried registers may also be referenced with these names. (See also *Referencing Subnodes*.)

**node name** The name given to a signal (wire) connecting two or more primitives. A node name may contain up to eight alphanumeric characters. Case is significant.

**pattern matching** Used to simplify entry of input vectors during simulation. User-specified characters are used in conjunction with wildcard characters, for example, to search for a particular group of node names or macrocells.

**Plot File (.WAV)** A Functional Simulator output file that contains a simple waveform description of the behavior of nodes requested with the PLOT command. These waveforms indicate High and Low Levels, High Impedance, Transitions, and Undefined. Plot Files may be printed at any time during or after a simulation session.

**predefined names** To allow complete access to all nodes within an EPLD, the Functional Simulator automatically creates node names for each element of BUSTER (EPB1400) buses, pins (except VCC and GND) and macrocells. Buses are given predefined names .B0 to .B7; macrocells are given predefined names .M1 to .M#, where # is a 1- or 2-digit number. Pins are given predefined names .P1 to .P#, where # is a 1- or 2-digit number. (EP1800G pin names also include a letter, e.g., .PA10.) These predefined names may be used in the same manner as user-defined node names and displayed with the SYMBOLS command.

**predefined vector sequences** Four predefined input vector sequences are available for simulation: the binary counting sequence, the rotating bit sequence, the gray code sequence, and the glitch generation sequence. (See *Predefined Input Vector Sequences* and the PATTERN command description.)

**Save File (.SAV)** An optional file that saves the current simulation instructions, allowing you to return later to a previously saved state and thus avoid repetition of simulation sessions.

**subnode** An internal node of an I/O primitive (Output Enable, Clear, Preset, Clock, Feedback, Input Vector, and D or T inputs). A subnode consists of the subnode name and a subnode extension. Refer to *Referencing Subnodes*.

**.TBL File** See **Watch File**.

**vector** A vector specifies the input logic levels for individual nodes within a design. The Functional Simulator uses vectors to simulate the behavior of the design. Vectors may be specified in .VEC files or defined with the **PATTERN** command.

**vector clock** The unit of time during which a single vector is processed.

**Vector File (.VEC)** Contains vectors that specify the logic levels of nodes in an EPLD design, which the Functional Simulator uses to test its logical operation. This file is a “1’s and 0’s file” describing the input conditions.

**vector sequences** See **predefined vector sequences**.

**Virtual Logic Analyzer (VLA)** Lets you view and analyze all input and output waveforms of your design in an interactive manner. The VLA allows you to view and manipulate the graphical display of the simulation in three independent windows, and may be invoked at any time during or after a simulation session.

**.WAV File** See **Plot File**.

**Watch File (.TBL)** A Functional Simulator output file that contains a tabular-format description (in 1’s and 0’s) of the state of the nodes requested by the **WATCH** command. Watch Files may be printed at any time during or after a simulation session. (Refer to the description of the **WATCH** command in *Command Reference*.)

**wildcard characters** Provide a shorthand notation for pattern matching of node names during simulation. The asterisk (\*) represents any string of characters; the question mark (?) represents any single character.



# Index

---

This index covers the *A+PLUS User Guide*, *A+PLUS Reference Guide*, and the *State Machine Entry* and *Functional Simulator* options.

Page number prefixes refer to the following sections:

|                |                                  |
|----------------|----------------------------------|
| <b>U</b>       | User Guide                       |
| <b>UBE</b>     | Boolean Equation Entry           |
| <b>USM</b>     | State Machine Entry              |
| <b>U(SM)M</b>  | State Machine Converter Messages |
| <b>UFS</b>     | Functional Simulator             |
| <b>ULA</b>     | Virtual Logic Analyzer           |
| <b>U(FS)M</b>  | Functional Simulator Messages    |
| <br>           |                                  |
| <b>R</b>       | Reference Guide                  |
| <b>RM</b>      | A+PLUS Messages                  |
| <b>RA – RF</b> | Appendixes A through F           |

## A

### A+PLUS

- A+PLUS messages
  - saving in ADP.LOG, **R3-10**
  - (*see* Error messages, Information messages, Warning messages)

#### APLUS Menu

- detailed description, **R3-2-4**
- executing DOS commands from, **U3-3, R3-4**
- exiting, **U3-3, R3-3**
- Help function, **U3-3, R3-3**
- invoking design editor from, **U3-3, R3-3**
- invoking Functional Simulator, **U3-3, R3-4**
- invoking LogicMap II, **U3-3, R3-3**
- invoking, **U3-2, R3-2**
- listing DOS directory from, **U3-3, R3-4**

- deinstallation, **U2-13**

- disk space requirements, **U2-9**

- hardware, (*see also* Logic Programmer)

- hardware installation, (*see* **LogicMap II** manual)

- hardware requirements, **U1-3**

- memory requirements, **U1-3**

- overview, **U1-2**

- software installation, (*see* Installation)

- updates, **U2-2, RE-4**

### Active Low signals

- alternate notation, **UBE-31**

- predefined active low signals, **UBE-31**

- recommended notation, **UBE-31**

- ADF-to-LEF translation, (*see* Altera Design Processor)

### ADLIB, **R1-4**

- ADLIB** distribution diskette, **U2-3**

- part field of Title Block, **R1-4**

- ADP, (*see* Altera Design Processor)

### ADP.LOG, **U1-9, R3-10**

- Algorithmic State Machine chart, **USM-8**

- Altera Design File, **U1-6, R2-2**

- comments

  - BNF syntax, **R2-3**

  - delimiters, **R2-3**

  - legal characters, **R2-3**

- Declarations Section, **UBE-17**

  - BNF syntax, **R2-6**

  - Inputs Section, **UBE-19, R2-6**

- legal pin name characters, **UBE-19**
  - restrictions, **UBE-19**
- Options Section
  - legal and default option values, **UBE-18, R2-6**
    - (*see also* Turbo-Bit, Security Bit)
- Outputs Section, **UBE-20, R2-6**
  - bidirectional pins, **UBE-20**
  - buried register outputs, **UBE-20**
- Part Section, **R2-6**
  - automatic part selection, **UBE-18, R2-6, R2-7**
  - with **MACRO** and **ADLIB**, **R2-6**
- End Statement, **UBE-34, R2-10**
- Equations Section, **UBE-32, R2-9**
  - BNF syntax, **R2-9**
- full BNF syntax, **R2-11-12**
- functions, **R2-2**
- Header Section, **UBE-17, R2-4**
  - BNF syntax, **R2-5**
  - character count in fields, **R2-5**
  - legal characters, **R2-4**
- keywords, **R2-3**
  - white space before, **R2-3**
- Network Section, **UBE-20**
  - active low signals
    - conventions, **UBE-31**
    - predefined, **UBE-31**
  - BNF syntax, **R2-8**
  - buried register outputs, **UBE-23**
  - clocking, (*see* Clock signal)
  - inputs to Bus I/O primitives, **UBE-21**
  - inputs to I/O primitives, **UBE-21**
  - legal node name characters, **UBE-30**
  - mnemonic primitive syntax, **UBE-24**
  - pin and node naming conventions, **UBE-27–29**
- submitting to ADP, **U3-5, R3-7**
- white space, **R2-3**
- Altera Design Processor
  - ADP Menu
    - default processing options, **U3-4, R3-5**
    - detailed description, **R3-5–9**
    - Execute function, **U3-6, R3-9**
    - exiting from, **U3-5, R3-6**
    - File Name(s) function, **U3-5, R3-7**
    - Help function, **U3-5, R3-6**
    - Input Format function, **U3-5, R3-7**

## Altera Design Processor – *continued*

- Inversion Control function, **U1-11**, **U3-5**, **R3-8**
- invoking, **U3-3**, **U3-4**, **R3-3**, **R3-5**
- LEF Analyzer function, **U3-5**, **R3-8**
- Minimizer function, **U3-5**, **R3-7**
- returning to APLUS Menu, **U3-5**
- Assembler
  - Assembler module functions, **U1-12**
  - combining multiple files, **R3-7**
  - default menu options, **U3-4**, **R3-5**
  - executing, **U3-6**, **R3-9**
- Expander
  - Expander module functions, **U1-10**
- Fitter, **R4-1–2**
  - Fitter module functions, **U1-11**, **R4-1**
- Flattener
  - Flattener module functions, **U1-9**
  - Inversion Control function, **U1-11**, **U3-5**, **R3-8**
  - invoking from DOS (stand-alone mode), **R3-11**, **R3-12**
- LEF Analyzer
  - LEF Analyzer module functions, **U1-11**, **R3-8**, **R3-3**
  - output file contents, **R3-8**, **U1-11**
- Minimizer
  - Minimizer module functions, **U1-11**, **U3-5**, **R3-7**
  - summary of automatic functions, **R3-10**
- Translator
  - ADF-to-LEF translation, **R3-1**
  - automatic part selection, **U1-10**
  - Translator module functions, **U1-10**, **R3-1**
- Altera Primitive Library, **U1-5**
  - foldout pages, **R3-3–5**
  - (*see also* Primitives)
- Altera Programmable Logic User System, (*see* A+PLUS)
- ALTERA.SYM, **R3-13**
- AND operators, (*see* Boolean equations)
- AND2 through AND12, **R1-9**
- APLUS Menu, (*see* A+PLUS)
- Ascii file-transfer protocol, **R3-10**
- ASM chart, **U3-8**
- asterisks
  - as repeat factors in FSIM vector patterns, **U3-13**
  - in Header Section, **U3-47**, **R2-4**
  - in JEDEC files, **R2-2**



- in Macrocell Interconnection Cross Reference, **R4-6**
- in pin names, **UBE-19, USM-49, R2-7**  
(*see also* pattern matching)
- asynchronous clocks, (*see* Clock signal)
  - asynchronous clock buffer, (*see* CLKB)
- AUTO**, (*see* automatic part selection)
- AUTOEXEC.BAK**, **U2-12**
- AUTOEXEC.BAT**, **U2-12**
- automatic part selection, **U1-10, UBE-18, USM-11, USM-48, R2-6, R3-10**
  - restrictions, **UBE-18, USM-11, USM-48, R2-7**

## **B**

- backups
  - of distribution diskettes, **U2-8**
- Backus-Naur Form
  - notation rules, **RC-1**  
(*see also* BNF syntax)
- BAND** primitives, **R1-14**
- BBUF**, **R1-10**
- BEGIN** command, (*see* FSIM commands)
- BEVDIS** sample design, **UBE-5-14, UFS-17**
  - BEVDIS.LEF**, **RB-4**
  - BEVDIS.RPT**, **R4-8**
- bidirectional pins, **UBE-20**
  - simulating in FSIM, **UFS-49**
- BIN**, (*see* FSIM commands: **GROUP**)
- binary base, (*see* Functional Simulator: groups: number bases)
- binary counting sequence, (*see* Functional Simulator)
- blank spaces, (*see* white space)
- BNAND** primitives, **R1-16**
- BNF syntax
  - for Altera Design File, **R2-11-12**
  - for State Machine File, **USM-58-60**
- BNOR** primitives, **R1-9**
- Boolean Equation Entry, **U1-8**
  - additional guidelines, **UBE-35**
  - functional description, **UBE-3**
  - general requirements, **UBE-2**
  - sample session, **UBE-5-14**
  - submitting ADFs to ADP, **U3-5, R3-7**
- Boolean equations
  - created during ADF-to-LEF translation, **RB-1**
  - expansion during design processing, **U1-10**

Boolean equations – *continued*

- permitted operators, **UBE-32**
- using intermediate equations, **UBE-33**  
(*see also* Altera Design File: Equations Section)
- Boolean expressions
  - in schematic drawings, **USM-51**
  - in State Machine Files, **USM-54**
- BOR** primitives, **R1-13**
- BREAK** command, (*see* FSIM commands)
- breakpoints, (*see* Functional Simulator)
- bubble gates, (*see* **BAND**, **BNAND**, **BNOR**, and **BOR**)
- Buried Registers Section, (*see* Utilization Report)
- buried macrocells
  - assigning to pins, **UBE-23**
  - EPLDs supporting, **UBE-23**
- buried pins, **R4-6**
- buried registers, **R4-3**
  - buried register outputs, **UBE-20**, **UBE-23**
- Bus I/O primitives, **R1-46-59**
  - BUSX**, **UBE-21**, **R1-48**
  - general rules, **R1-46**
  - LBUSI**, **UBE-21**, **R1-50**
  - LBUSO**, **UBE-21**, **R1-52**
  - LINP8**, **UBE-21**, **R1-54**
  - naming conventions, **UBE-28**
  - RBUSI**, **UBE-21**, **R1-56**
  - RINP8**, **UBE-21**, **R1-58**
- bus port pins
  - simulating in FSIM, **UFS-50**
  - using predefined node names in FSIM, **UFS-51**
- Bus transceiver, (*see* **BUSX**)
- BUSTER** (EPB1400)
  - block diagram, **RA-31**
  - buried macrocells, **UBE-23**
  - clocking diagrams, **RA-26**
  - combining internal bus nodes in VLA, **ULA-23**
  - dual I/O feedback, **UBE-22**
  - grouping internal bus nodes in FSIM, **UFS-70**
  - inputs to bus primitives, **UBE-21**
  - macrocell group table, **RA-30**
  - part description, **RA-24**
  - number of bus primitives allowed, **R1-46**
  - restrictions with LogicMap II, **U1-14**

signal reevaluation in FSIM vector processing, *UFS-42*  
simulating buses, (*see also* Virtual Logic Analyzer)  
**BUSX**, *UBE-21*, *R1-46*, 48  
setting control signals in FSIM, *UFS-50*

## **C**

carriage returns, (*see* white space)  
CASE statements, (*see* transitions)  
characters  
    legal characters for  
        active low signals, *UBE-31*  
        node names, *UBE-30*  
        comments, *R2-3*  
        FSIM Vector Files, *UFS-12*  
        Header Section, *USM-47*, *R2-4*  
        JEDEC files, *RD-4*  
        pin names, *UBE-30*, *USM-49*  
        state machine clear name, *USM-51*  
        state machine clock name, *USM-51*  
        state machine names, *USM-42*  
        state names, *USM-52*  
        state variable names, *USM-52*, *USM-53*  
        state variable values, *USM-52*  
        truth tables, *USM-56*  
    reserved words in FSIM, *UFS-53*  
**CLEAR** command, (*see* FSIM commands)  
Clear signal, *UBE-33*, *USM-13*  
    effects on vector processing in FSIM, *UFS-42*  
    inputs to primitives, *UBE-21*  
    in state machines, *USM-40*, *USM-51*  
    in Utilization Report, *R4-5*  
    legal characters, *USM-51*  
Clear subsection, (*see* State Machine File)  
**CLKB**, *R1-11*  
    (*see also* Clock signal: asynchronous)  
Clock signal, *UBE-33*  
    asynchronous, *USM-13*  
        logic-driven, *UBE-22*, *USM-40*, *USM-51*  
        pin-driven, *UBE-21*  
    clock groups in Utilization Report, *R4-5*  
    effects on vector processing in FSIM, *UFS-42*  
    inputs to bus I/O primitives, *UBE-21*  
    legal characters, *USM-51*  
    simulating designs with externally connected pins, *UFS-51*

Clock signal – *continued*

- synchronous, **UBE-21, USM-13, USM-40, USM-51**
- Clock subsection, (*see* State Machine File)
- clocking diagrams, (*see* Appendix A)
- CMD file, (*see* Functional Simulator: Command File)
- CMP files, (*see* P-CAD)
- COCF, **R1-24**
- COIF, **UBE-21, R1-25**
- COLF, **R1-26**
- combinatorial feedback, **UBE-33, UBE-35**
- combining files
  - entering filenames at ADP menu prompts, **R3-7**
  - with different formats, **R3-11**
- comments
  - in Logic Equation File, **RB-3**  
(*see also* Altera Design File, State Machine File)
- conditional outputs, (*see* Output: state machine outputs)
- conditional transitions, (*see* transitions)
- CONF, **R1-27**
  - used for reserving unused pins **R4-3**
- CONFIG.BAK, **U2-12**
- CONFIG.SYS, **U2-12**
- CONTINUE command, (*see* FSIM commands)
- convergence limit, (*see* Functional Simulator)
- converters, (*see* State Machine Entry and FutureNet DASH)
- copy protection, **U2-8**
- CORF, **R1-28**
- Crosstalk, **RE-10**
- CYCLE command, (*see* FSIM commands)

## D

- DASH, (*see* FutureNet DASH)
- data buffer, (*see* Virtual Logic Analyzer)
- De Morgan's inversion, (*see* Altera Design Processor: Inversion Control)
- DEC, (*see* FSIM commands: GROUP)
- decimal base, (*see* Functional Simulator: groups: number bases)
- Declarations Section, (*see* Altera Design File, State Machine File)
- Deinstallation, **U2-13**
  - De-Installation Menu, **U2-13**
- DESCRIBE command, (*see* FSIM commands)

- design entry methods, (*see* State Machine Entry, LogiCaps, FutureNet DASH, P-CAD, Netlist Entry, Boolean Equation Entry)
- DIP packages, **RA-2**
- disk drives
  - 360 Kbyte vs. 1.2 Mbyte, **U2-9**
- disk space
  - required for A+PLUS, **U2-9**
- DISPLAY** command, (*see* FSIM commands)
- distribution diskettes
  - backups, **U2-8**
  - for installation, **U2-3**
- DOS
  - executing DOS commands
    - from A+PLUS, **U3-3, R3-4**
    - from FSIM, **UFS-66**
  - invoking ADP from (stand-alone mode), **R3-12**
  - invoking FCV from (stand-alone mode), **R3-12**
  - invoking LogiCaps from (stand-alone mode), **R3-13**
  - invoking LogicMap II from (stand-alone mode), **R3-14**
  - invoking State Machine Converter from (stand-alone mode), **R3-14**
  - invoking FSIM from, **R3-12**
    - batch mode, **UFS-38**
    - interactive mode, **UFS-19**
  - listing DOS files while in A+PLUS, **U3-3, R3-4**
  - recommended version, **U2-3**
- downloading files, (*see* files)
- dual I/O feedback, **UBE-20, UBE-22, R4-4**
  - naming conventions, **UBE-27**

## **E**

- ECF**, (*see* LogicMap II)
- ECHO** command, (*see* FSIM commands)
- editors
  - FutureNet DASH, **U1-7**
  - LogiCaps, **U1-7**
- Electronic Application Briefs, (*see* Appendix E)
- Electronic Application Utilities, (*see* Appendix E)
- Electronic Design Support Service, **RE-1**
- End Statement, (*see* Altera Design File, State Machine File)
- ENDCASE**, (*see* transitions)
- EP310**
  - block diagram, **RA-6**
  - clocking diagrams, **RA-4**

EP310 – *continued*

macrocell group table, **RA-5**  
part description, **RA-3**

EP320

block diagram, **RA-7**  
clocking diagrams, **RA-4**  
macrocell group table, **RA-5**  
part description, **RA-3**

EP600/610

block diagram, **RA-11**  
clocking diagrams, **RA-9**  
macrocell group table, **RA-10**  
part description, **RA-8**

EP900/910

block diagram, **RA-15**  
clocking diagrams, **RA-13**  
macrocell group table, **RA-14**  
part description, **RA-12**

EP1210

block diagram, **RA-23**  
clocking diagrams, **RA-17**  
macrocell group table, **RA-21**  
part description, **RA-16**

EP1800

block diagram, **RA-38**  
buried macrocells, **UBE-23**  
clocking diagrams, **RA-34**  
dual I/O feedback, **UBE-22**  
macrocell group table, **RA-36**  
part description, **RA-33**

EPB1400, (*see* BUSTER)

EPLD.SYS, **U2-10**

EPLDs

general information, **RA-2**  
package configurations, **RA-2**  
power-up state, **USM-41**  
programming and verifying, (*see* LogicMap II manual)  
simulating, (*see* Functional Simulator)EPLDs

Equation primitives, **R1-20-22**

**EQN1**, **R1-21**

**EQN8**, **R1-22**

Equations Section, (*see also* Altera Design File or State Machine File)

- equations, (*see* Boolean equations)
- Error messages
  - A+PLUS, **R3-10**
  - Functional Simulator, **U(FS)M-2**
  - State Machine Converter, **U(SM)M-1**
- EXAMPLE1 sample design, **USM-7-19**
- EXAMPLE2 sample design, **USM-20-28**
- EXAMPLE3 sample design, **USM-29-39**
- exclamation point
  - after pin numbers, **R4-4**
  - after primitive names, **R4-4**
- exclusive NOR, (*see* XNOR)
- exclusive OR, (*see* XOR)
- EXECUTE command, (*see* FSIM commands)
- executing the ADP, **U3-6, R3-9**
- exiting
  - from ADP Menu, **U3-5, R3-6**
  - from APLUS Menu, **U3-3, R3-3**
  - from Functional Simulator, **UFS-77**
  - from temporary DOS environment, **U3-3, R3-4**
  - from VLA, **ULA-24**
- Extended Software Warranty, **RE-3**

## F

- FCV, (*see* FutureNet DASH)
- Feedback
  - combinatorial, **UBE-33, UBE-35**  
(*see also* COCF, NOCF, ROCF)
  - dual I/O, **R4-4**
  - EPLDs supporting, **UBE-22**
  - feedback group in Utilization Report, **R4-5**
  - I/O, (*see* COIF, ROIF, TOIF)
  - JK, (*see* JOJF, NOJF)
  - latched, (*see* COLF, ROLF)
  - local, **R4-6**
  - naming conventions, **UBE-27**
  - registered, (*see* CORF, NORF, RORF)
  - SR, (*see* NOSF, SOSF)
  - T, (*see* NOTF, TOTF)
- Fido, (*see* Appendix E)
- filename extensions
  - specifying in ADP Menu, **U3-5, R3-7**

## files

- analyzing, **U1-11**, **U3-5**, **R3-8**
- combining, **R3-7**, **R3-11**
- converting PIN files into ADFs, **U1-7**, **R3-12**
- converting SMFs into ADFs, **U1-7**, **R3-14**
- downloading from Altera, **RE-12**
- editing from A+PLUS, **U3-3**, **R3-3**
- file-transfer protocols, **RE-10**
- minimizing, **U1-11**, **R3-7**
- Netlist format, **U1-6**
- partitioning large files, **R3-9**
- submitting to ADP, **U3-5**, **R3-7**
- uploading to Altera, **RE-10**

Fitter, (*see* Altera Design Processor, Utilization Report)

**FNET**, (*see* FutureNet DASH)

**FORCE** command, (*see* FSIM commands)

FSIM, (*see* Functional Simulator)

### FSIM commands

**BEGIN**, **UFS-58**

**BREAK**, **UFS-59**

effect of cycle length, **UFS-45**

nested commands, **UFS-27**

specifying cycle value, **UFS-56**

use of semicolon, **UFS-54**

**CLEAR**, **UFS-61**

command format, **UFS-55**, **UFS-56**

node list, **UFS-55**

node value list, **UFS-55**

pathnames, **UFS-56**

**CONTINUE**, **UFS-62**

**CYCLE**, **UFS-42**, **UFS-63**

defining cycle value, **UFS-26**

effect on predefined vector sequences, **UFS-47**

expanding signal lengths, **UFS-44**, **UFS-45**

effect on FSIM prompt, **UFS-23**

**DESCRIBE**, **UFS-64**

**DISPLAY**, **UFS-65**

**DOS**, **UFS-66**

**ECHO**, **UFS-67**

**EXECUTE**, **UFS-68**

**FORCE**, **UFS-69**

usage with **BREAK**, **UFS-69**

general description, **UFS-8–10**



FSIM commands – *continued*

**GROUP, UFS-70**

HEX, DEC, OCT and BIN number bases, *UFS-70*  
predefined node names, *UFS-51*

**HELP, UFS-71**

**INIT, UFS-72**

**LOG, UFS-73**

**PATTERN, UFS-74**

nested patterns, *UFS-13*  
pattern format, *UFS-12*  
patterns for bidirectional pins, *UFS-49*  
patterns for bus port pins, *UFS-50*  
patterns for externally connected clock pins, *UFS-51*  
predefined vector sequences, *UFS-46–47*  
repeat factor, *UFS-13*

**PLOT, UFS-75**

use with VLA, *ULA-3*  
restrictions, *UFS-70*

**QUIT, UFS-77**

**RESTORE, UFS-78**

**SAVE, UFS-79**

**SIMULATE, UFS-80**

effect of cycle length, *UFS-45*  
entering cycle values, *UFS-29, UFS-56*

**STATUS, UFS-81**

**SYMBOLS, UFS-82**

displaying predefined node names, *UFS-51*  
pattern matching, *UFS-24*

**VECTOR, UFS-83**

with groups of nodes, *UFS-12*

**VIEW, UFS-84**

(*see also* Virtual Logic Analyzer)

**WATCH, UFS-85**

**GROUP**

use with VECTOR command, *UFS-12*

**PATTERN**

Functional Simulator, *U1-13*

(*see also* FSIM Commands, Virtual Logic Analyzer)

batch mode, *UFS-37*

command termination, *UFS-54, UFS-56*

line numbers in error messages, *U(FS)M-1*

usage with VLA, *ULA-4*

bidirectional pins, *UFS-49*

- INP node name extension, *UFS-49*
- breakpoints, *UFS-9, UFS-27*
  - clearing, *UFS-61*
  - highlighting in VLA, *ULA-22*
  - setting, *UFS-59*
  - use of FORCE command, *UFS-69*
- bus port pins, *UFS-50*
- Command File, *UFS-3, UFS-14*
  - executing, *UFS-68*
  - command termination, *UFS-54, UFS-56*
- convergence limit, *UFS-42*
- cycle value, *UFS-29, UFS-56*
  - relative vs absolute, *UFS-56*
  - setting, *UFS-63*
  - specifying, *UFS-80*
- displaying files, *UFS-65*
- displaying simulation status, *UFS-81*
- entering vector patterns, *UFS-74*
- Error messages, *U(FS)M-2*
- exiting, *UFS-77*
- expanding signal lengths, *UFS-43*
- externally connected clock pins, *UFS-51*
- FSIM distribution diskette, *U2-4*
- functional description, *UFS-3-5*
- groups
  - most and least significant bits, *UFS-70*
  - number bases, *UFS-12, UFS-14, UFS-70*
- high impedance, *UFS-12*
  - use when simulating bidirectional pins, *UFS-49*
  - use when simulating bus port pins, *UFS-50*
- interactive mode, *UFS-19*
  - command termination, *UFS-54, UFS-56*
- invoking
  - from APLUS Menu, *U3-3, R3-4, UFS-19, UFS-38*
  - from DOS (stand-alone mode), *R3-12, UFS-19, UFS-38*
- invoking Virtual Logic Analyzer, *UFS-84*
- Log File, *UFS-3, UFS-15*
  - creating, *UFS-73*
- nested commands, *UFS-27*
- nodes
  - displaying node names, *UFS-82*
  - grouping, *UFS-70*
  - internal subnodes of I/O primitives, *UFS-52*
  - node list, *UFS-55*
  - node value list, *UFS-55*

## Functional Simulator: nodes – *continued*

- predefined node names
  - buses, *UFS-51*
  - macrocells, *UFS-51*
  - pins, *UFS-51*
- pathnames, *UFS-56*
- pattern matching, *UFS-24*, *UFS-85*
- Plot File, *UFS-3*, *UFS-15*
  - creating, *UFS-75*
  - resetting, *UFS-58*
  - usage with VLA, *ULA-3*
- prompt
  - effect of **CYCLE** command, *UFS-23*
  - format, *UFS-23*
- predefined vector sequences
  - binary counting, *UFS-46*
  - effect of cycle value, *UFS-47*
  - glitch generator, *UFS-47*
  - gray code, *UFS-47*
  - rotating bit, *UFS-46*
- propagation of undefined logic levels, *UFS-48*
- reserved words, *UFS-53*
- restoring previous simulation environment, *UFS-16*, *UFS-78*
- resuming simulation, *UFS-62*
- sample sessions
  - batch mode, *UFS-37*
  - interactive mode, *UFS-17*
- Save File, *UFS-16*
  - creating, *UFS-79*
  - restoring, *UFS-78*
- screen displays, *UFS-30*
- simulation cover percentage, *UFS-16*, *UFS-76*, *UFS-81*, *UFS-85*
- vectors
  - specifying input vectors, *UFS-83*
  - vector clock
    - definition, *UFS-42*
  - vector cycles, *UFS-10*
  - vector patterns
    - for bidirectional pins, *UFS-49*
    - for bus port pins, *UFS-50*
    - for externally connected clock pins, *UFS-51*
    - repeat factor, *UFS-13*

- predefined vector sequences, **UFS-46**
- Vector File, **UFS-3, UFS-11**
  - comments in, **UFS-21**
  - expanding signal lengths, **UFS-43**
  - legal logic level characters, **UFS-12**
  - nested patterns, **UFS-13**
  - number bases, **UFS-14**
  - overriding vectors in, **UFS-13, UFS-69**
  - PATTERN format, **UFS-12**
  - switching files, **UFS-11**
  - TABLE format, **UFS-13**
  - usage with groups of nodes, **UFS-12**
- vector processing sequence, **UFS-42**
- Warning messages, **U(FS)M-12**
- Watch File, **UFS-3, UFS-15**
  - creating, **UFS-85**
- FutureNet DASH, **U1-7**
  - primitives, **R1-2**
  - FNET** distribution diskette, **U2-4**
  - invoking from APLUS Menu, **U3-3, R3-3**
  - Pinlist Converter, **R2-2**
    - invoking from DOS (stand-alone mode), **R3-11, R3-12**
  - submitting PIN files to ADP, **U3-5, R3-7**

## G

- GND** (ground), **R1-12**
- glitch generator sequence, (*see* Functional Simulator)
- gray code sequence, (*see* Functional Simulator)
- GROUP** command, (*see* FSIM commands)

## H

- hard disk
  - installing A+PLUS software on, **U2-9**
- hardware installation, (*see* LogicMap II manual)
- Header Section
  - of Logic Equation File, **R3-3**
  - of Utilization Report, **R4-3**
  - (*see also* Altera Design File, State Machine File)
- help
  - invoking in ADP Menu, **R3-6**
  - invoking in APLUS Menu, **U3-3, R3-3**
  - invoking in Functional Simulator, **UFS-71**
  - invoking in Virtual Logic Analyzer, **ULA-18**

**HEX**, (*see* FSIM commands: **GROUP**)  
hexadecimal base, (*see* Functional Simulator: groups: number bases)  
high impedance, **UFS-12**  
    use when simulating bidirectional pins, **UFS-49**, **UFS-50**  
    (*see also* Virtual Logic Analyzer)

## I

I/O primitives, **R1-23**, **R1-45**  
    ADF-to-LEF translation of, **RB-3**  
    **COCF**, **R1-24**  
    **COIF**, **R1-25**  
    **COLF**, **R1-26**  
    **CONF**, **R1-27**  
    **CORF**, **R1-28**  
    internal subnodes of, **UFS-52**  
    **JOJF**, **R1-29**  
    **JONF**, **R1-30**  
    mnemonic names for, **R1-23**  
    **NOCF**, **R1-31**  
    **NOJF**, **R1-32**  
    **NORF**, **R1-33**  
    **NOSE**, **R1-34**  
    **NOTE**, **R1-35**  
    pin and node naming conventions, **UBE-27**  
    **ROCF**, **R1-36**  
    **ROIF**, **R1-37**  
    **ROLF**, **R1-38**  
    **RONF**, **R1-39**  
    **RORF**, **R1-40**  
    **SONF**, **R1-41**  
    **SOSF**, **R1-42**  
    **TOIF**, **R1-43**  
    **TONF**, **R1-44**  
    **TOTF**, **R1-45**  
**IF-THEN** statements, (*see* transitions: conditional)  
Information messages  
    **A+PLUS**, **R3-10**  
    State Machine Converter, **U(SM)M-11**  
**INITIALIZE** command, (*see* FSIM commands)  
**INP**, **R1-6**  
**.INP** pin name extension (*see* Functional Simulator: bidirectional pins)  
Input primitives  
    **INP**, **R1-6**  
    **LINP**, **R1-7**

- pin and node naming conventions, **UBE-27**
- input format
  - specifying in ADP Menu, **U3-5, R3-7**
- input vectors, (*see* Functional Simulator)
- Inputs Section, (*see* Altera Design File, State Machine File)
- Installation
  - AUTOEXEC.BAT** file, **U2-12**
  - CONFIG.SYS** file, **U2-12**
  - distribution diskettes, **U2-3**
  - EPLD.SYS** file, **U2-10**
  - hardware, (*see* LogicMap II manual)
  - installation checklist, **U2-2**
  - Installation Menu, **U2-11**
  - installation procedure, **U2-9**
  - Main Menu, **U2-10**
  - making backup diskettes, **U2-8**
  - READ.ME** file, **U2-2**
  - recommended DOS version, **U2-3**
- intermediate equations, (*see* Boolean equations)
- intermediate LEFs, (*see* Logic Equation File)
- internal subnodes, (*see* nodes)
- Inversion control (*see* Altera Design Processor: Inversion Control)

## **J**

- J-lead packages, **RA-2**
- JEDEC File, **UFS-11**
  - character substitution in, **UBE-19, R2-7**
  - description of format, **RD-1**
  - editing and viewing, (*see* LogicMap II)
  - legal characters in, **RD-4**
- JOJF**, **R1-29**
- JONF**, **R1-30**

## **K**

- Kermit, **RE-10**
- Keyword state variable format, **USM-53**
- keywords, (*see* Altera Design File, State Machine File, Functional Simulator)

## L

Latch Enable signal, **UBE-33**

latches

    cross-coupled, **UBE-33**

    propagation of undefined logic levels, **UFS-48**

    (*see also* Utilization Report)

Latches Section, (*see* Utilization Report)

**LBUSI, R1-46, R1-50**

**LBUSO, R1-46, R1-52**

least significant bit, **UFS-70**

LEF, (*see* Logic Equation File, Altera Design Processor: LEF Analyzer)

LEF Analyzer, (*see* Altera Design Processor)

legal characters, (*see* characters)

line feeds, (*see* white space)

**LINP, R1-7**

**LINP8, R1-46, R1-54**

local feedback, **R4-6**

LOG file, (*see* Functional Simulator: Log File)

LOGFILE command, (*see* FSIM commands)

logging on

    to Fido, **RE-3**

Logic Analyzer, (*see* Logic Equation File and Altera Design Processor:  
LEF Analyzer)

Logic Equation File, **U1-10, RB-1**

    ADF-to-LEF translation, **U1-10**

    analyzing, **U1-11, U3-5, R3-8**

    comments in, **RB-3**

    converting, **U1-11**

    intermediate, **U1-10, U1-11, R3-8**

    LEF Header Section, **RB-3**

logic inversion, (*see* Altera Design Processor: Inversion Control)

logic levels

    legal characters, **UFS-12**

Logic primitives, **R1-8-19**

    ADF-to-LEF translation, **RB-1**

**AND2** through **AND12, R1-9**

**BAND2** through **BAND12, R1-14**

**BBUF, R1-10**

**BNAND2** through **BNAND12, R1-16**

**BNOR2** through **BNOR12, R1-9**

**BOR2** through **BOR12, R1-13**

**CLKB, R1-11**

**GND, R1-12**

**NAND2** through **NAND12, R1-13**

## Logic primitives – *continued*

- NOR2 through NOR12, **R1-14**
- NOT, **R1-15**
- OR2 through OR12, **R1-16**
- VCC, **R1-17**
- XNOR, **R1-18**
- XOR, **R1-19**
- Logic Programmer
  - programming card, **U1-3, R3-3**
    - default address, **U2-11**
    - (*see also* LogicMap II manual)
  - programming unit, **U1-2, U1-14**
- logic reduction, (*see* Altera Design Processor: Minimizer)
- LogiCaps, **U1-7**
  - invoking from APLUS Menu, **U3-3, R3-3**
  - invoking from DOS (stand-alone mode), **R3-13**
  - LOGICAPS distribution diskette, **U2-3**
  - Symbol Numbers command, **R1-2**
  - UTILITIES distribution diskette, **U2-3**
- LOGICAPS.CFG, **R3-14**
- LogicMap II, **U1-14**
  - ECF distribution diskette, **U2-3**
  - invoking from APLUS Menu, **U3-3, R3-3**
  - invoking from DOS (stand-alone mode), **R3-14**
  - LOGICMAP distribution diskette, **U2-3**
  - toggleing Turbo-Bit and Security Bit, **UBE-35, USM-41, R2-6**

## M

- Machine Section, (*see* State Machine File)
- Macrocells, (*see also* Utilization Report)
  - buried, **UBE-23, R4-6**
  - clear signal, **R4-5**
  - clock signal, **R4-5**
  - feedback group, **R4-5**
  - macrocell group assignments, (*see* Appendix A)
  - Macrocell Interconnection Cross Reference, (*see* Utilization Report)
  - output enable signal, **R4-5**
  - predefined node names in FSIM, **UFS-51**
  - preset signal, **R4-5**
- MacroFunctions, **R3-10**
  - flattening, **U1-9, R3-10**



- MACROLIB** distribution diskette, *U2-3*
- MACROLIB-TTL** distribution diskette, *U2-3*
- Standard, *U1-5*
- TTL, *U1-5*
- memory
  - CONFIG.SYS** file requirements, *U2-12*
  - increasing for larger designs, *R3-11*
  - potential problems, *R3-4*
  - requirements for running A+PLUS, *U1-3*
- Minimizer, (*see* Altera Design Processor)
- Minitel, *RE-10*
- mnemonic names
  - for pins and nodes, *UBE-24*
  - for I/O primitives, *R1-23*
- modem
  - Fido communication parameters, *RE-2*
  - link to Altera, *RE-1*
- Modem7, *RE-10*
- most significant bit, *UFS-70*
- multiple files
  - combining, *R3-7*

**N**

- N.C. (*see* pins: with no internal connections)
- naming conventions (*see* Altera Design File: Network Section)
- NAND2** through **NAND12**, *R1-13*
- nested patterns, (*see* Functional Simulator)
- Netlist entry method, *U1-8, R2-13*
  - submitting ADFs to ADP, *U3-5, R3-7*
- Network Section, (*see* Altera Design File or State Machine File)
- NOCF**, *R1-31*
  - use with asynchronous clocking, *UBE-22*
- nodes
  - displaying in FSIM, *UFS-82*
  - grouping in FSIM, *UFS-12, UFS-70*
  - internal subnodes of I/O primitives, *UFS-52*
  - propagation of undefined node levels, *UFS-48*
  - referencing predefined node names in FSIM, *UFS-51*
  - use of .INP node name extension in FSIM, *UFS-49*
  - legal node name characters, *UBE-30*
  - mnemonic node naming conventions, *UBE-27-28*
  - (*see also* FSIM commands: **BREAK**; Virtual Logic Analyzer)
- NOJF**, *R1-32*
- NOR2** through **NOR12**, *R1-14*

**NORF**, *R1-33*  
**NOSF**, *R1-34*  
**NOT**, *R1-15*  
**NOT operators**, (*see* Boolean equations)  
**NOTF**, *R1-35*

## **O**

**OCT**, (*see* FSIM commands: **GROUP**)  
octal base, (*see* Functional Simulator: groups: number bases)  
operators, (*see* Boolean equations)  
Options Section, (*see* Altera Design File, State Machine File)  
**OR operators**, (*see* Boolean equations)  
**OR2 through OR12**, *R1-16*  
Output  
    buried, *UBE-20*, *UBE-23*  
    Combinatorial, (*see* **COCF**, **COIF**, **COLF**, **CONF**, **CORF**)  
    **JK**, (*see* **JOJF**, **JONF**)  
    Registered, (*see* **ROCF**, **ROIF**, **ROLF**, **RONF**, **RORF**)  
    **SR**, (*see* **SONF**, **SOSF**)  
    state machine outputs, *USM-9*  
        as inputs to other state machines, *USM-40*  
        conditional output syntax, *USM-55*  
        restrictions, *USM-41*  
        unconditional output syntax, *USM-55*  
        use of auxiliary variables, *USM-41*  
    **T**, (*see* **TOIF**, **TONF**, **TOTF**)  
Output Enable signal, *UBE-33*  
    effects on vector processing in FSIM, *UFS-42*  
    in Utilization Report, *R4-5*  
    inputs to primitives, *UBE-21*  
Output Latch Enable signal, *UBE-33*  
    inputs to bus primitives, *UBE-21*  
Outputs Section, (*see* Altera Design File, State Machine File, Utilization Report)  
    difference from Outputs subsection in SMFs, *USM-51*  
Outputs subsection, (*see* State Machine File)

## **P**

**P-CAD**, *U1-6*  
    invoking from APLUS Menu, *U3-3*, *R3-3*  
    node name for GND, *R1-12*  
    node name for VCC, *R1-17*  
    submitting CMP files to ADP, *U3-5*, *R3-7*

- Part Section
  - automatic part selection, **R2-6**  
(*see also* Altera Design File, State Machine File)
- Part Utilization Section, (*see* Utilization Report)
- partitioning files, **R3-9**
- pathname
  - specifying in FSIM, **UFS-56**
- PATTERN** command, (*see* FSIM commands)
- pattern matching, **U3-3, R3-4**
  - in Functional Simulator, **UFS-24, UFS-85**
  - with input filenames, **R3-7**
- PC-CAPS, (*see* P-CAD)
- periods
  - in pin names, **UBE-19, USM-49, R2-7**
- PIN files, (*see* FutureNet DASH)
- Pin-grid array packages, **RA-2**
- Pin List Converter, (*see* FutureNet DASH)
- pins
  - bidirectional, **UBE-20, UBE-22**
    - simulating in FSIM, **UFS-49**
  - buried, **R4-6**
  - bus port pins
    - simulating in FSIM, **UFS-50**
  - dedicated input pins
    - /CRS, UBE-31**
    - /CWS, UBE-31**
  - in Macrocell Interconnection Cross Reference, **RA-6**
  - pin assignments
    - in Utilization Report, **RA-1, RA-4**
    - to buried macrocells, **UBE-23**
  - pin names
    - asterisks in, **UBE-19, USM-49, R2-7**
    - legal characters, **UBE-30, USM-49**
    - mnemonic names, **UBE-24**
    - naming conventions, **UBE-27**
    - periods in, **UBE-19, USM-49, R2-7**
    - tildes (~) in, **UBE-19, USM-49, R2-7**
  - pin numbers
    - notation used in macrocell group tables, **RA-2**
  - predefined node names in FSIM, **UFS-51**
  - reserved, **R4-3**
  - simulating externally connected clock pins, **UFS-51**
  - unused, **R4-3**
  - using I/O pins as input pins, **UBE-20**
  - with no internal connections (N.C.), **RA-3**

- PLCAD-SUPREME, **U2-5**
- PLCAD4, **U2-6**
- PLDS2, **U2-7**
- PLOT** command, (*see* FSIM commands)
- Positional state variable format, **USM-52**
- power-up state, **USM-20**
- predefined active low signals, **UBE-31**
- predefined node names, (*see* Functional Simulator)
- predefined vector sequences, (*see* Functional Simulator)
- Preset signal, **UBE-33**
  - effects on vector processing in FSIM, **UFS-42**
  - in EP310's, **RA-3**
  - inputs to primitives, **UBE-21**
  - in Utilization Report, **R4-5**
- Primitives
  - ADF-to-LEF translation, **RB-1-3**
  - bubble gates, (*see* **BAND**, **BNAND**, **BNOR**, and **BOR**)
  - bus I/O primitives, **R1-46-59**
  - equation primitives, **R1-20-22**
  - format description, **R1-2**
  - foldout pages, **RF-3-5**
  - followed by exclamation points, **R4-4**
  - I/O primitives, **R1-23**, **R1-45**
    - internal subnodes of, **UFS-52**
  - input primitives, **R1-5-7**
  - logic primitives, **R1-2**, **R1-8-19**
  - minimization of, **U1-11**
  - mnemonic names for, **R1-23**
  - multiple inputs, **R1-8**
  - node naming conventions, **UBE-27**
  - Title Block, **R1-3**
  - promotion of, **R4-4**
  - syntax with mnemonic pin and node names, **UBE-24**  
(*see also* Altera Primitive Library)
- printable characters, **USM-46**, **USM-47**, **R2-4**
- product terms
  - sharing, **UBE-35**  
(*see also* Utilization Report)
- promotion of primitives, **R3-8**

## Q

- question marks
  - in Macrocell Interconnection Cross Reference, **R4-6**  
(*see also* pattern matching)

## R

RAM, (*see* memory)  
RANGE, (*see* FSIM commands: **BREAK**)  
RBUSI, **R1-46**, **R1-56**  
Read Enable signal, **UBE-33**  
Read Strobe signal  
    active low logic, **UBE-31**  
    effects on vector processing in FSIM, **UFS-42**  
    inputs to bus primitives, **UBE-21**  
READ.ME file, **U2-2**  
Registers  
    buried, **R4-3**  
repeat factor  
    in FSIM vector patterns, **UFS-13**  
repeating processing, **R3-9**  
reserved pins, **R4-3**  
reserved words, (*see* Functional Simulator)  
resources, (*see* Utilization Report)  
RESTORE command, (*see* FSIM commands)  
RINP8, **R1-46**, **R1-58**  
ROCF, **R1-36**  
ROIF, **UBE-20**, **R1-37**  
ROLF, **R1-38**  
RONF, **R1-39**  
RORF, **R1-40**  
rotating bit, (*see* Functional Simulator)

## S

schematic design entry, (*see* LogiCaps, FutureNet DASH, P-CAD)  
SDF, **U1-9**, **R3-10**  
Security Bit, **UBE-35**, **USM-41**, **USM-42**, **R1-4**  
    default value, **UBE-18**, **R2-6**, **RB-3**  
    in JEDEC files, **RD-3**  
    toggling with LogicMap II, **UBE-35**, **USM-41**, **R2-6**  
semicolon (;)  
    use in FSIM commands, **UFS-54**  
    in nested **BREAK** commands, **UFS-27**  
serial numbers, **U2-2**  
SMV, (*see* State Machine Entry)  
Software Warranty, (*see* Extended Software Warranty)  
software installation, (*see* Installation)  
software update information, **RE-4**  
SONF, **R1-41**

- SOSF, *R1-42*
- stand-alone mode, *R3-11–14*
- State Machine Entry, *U1-7*
  - design guidelines, *USM-40-42*
  - functional description, *USM-3*
  - general requirements, *USM-2*
  - sample sessions
    - EXAMPLE1, *USM-7–19*
    - EXAMPLE2, *USM-20–28*
    - EXAMPLE3, *USM-29–39*
  - suggested design steps, *USM-5*
  - SMV distribution diskette, *U2-3*
  - State Machine Converter, *R2-2*
    - invoking from DOS (stand-alone mode), *R3-11, R3-14*
- State Machine File
  - comments
    - BNF syntax, *USM-46*
    - delimiters, *USM-46*
    - legal characters, *USM-46*
  - Declarations Section, *USM-47*
    - legal pin name characters, *USM-49*
  - Options Section, *USM-48*
    - legal and default option values, *USM-48*  
(*see also* Turbo-Bit, Security Bit)
  - Outputs Section, *USM-49*
    - multiple state machines, *USM-40*
    - use of state variables, *USM-40*
  - Part Section, *USM-48*
    - automatic part selection, *USM-48*
  - End Statement, *USM-57*
  - Equations Section, *USM-50*
    - guidelines, (*see* Altera Design File)
  - full BNF syntax, *USM-58-60*
  - Header Section, *USM-47*
    - character count in fields, *USM-47*
    - legal characters, *USM-47*
  - keywords *USM-46*
    - white space before *USM-46;*
  - Machine Section, *USM-50*
    - Clear subsection, *USM-51*
      - legal clear name characters, *USM-51*  
(*see also* Clear signal)
    - Clock subsection, *USM-51*
      - legal clock name characters, *USM-51*  
(*see also* Clock signal)

- legal state machine name characters, *USM-42, USM-51*
- Outputs subsection, *USM-41*
  - restrictions, *USM-41*
- States subsection, *USM-52*
  - Keyword state variable format, *USM-53*
  - legal state name characters, *USM-52*
  - legal state variable name characters, *USM-52*
  - legal state variable values, *USM-52*
  - Positional state variable format, *USM-52*
- Transitions and Outputs subsections, *USM-53*
  - CASE statement transitions, *USM-55*
  - conditional outputs, *USM-55*
  - conditional transitions, *USM-54*
  - unconditional outputs, *USM-55*
  - unconditional transitions, *USM-54*
- Network Section, *USM-50*
  - guidelines, (*see* Altera Design File)
  - submitting to ADP, *U3-5, R3-7*
- Truth Table Section, *USM-56*
  - definition of, *USM-10*
  - legal truth table values, *USM-56*
  - white space, *USM-46*
- state diagram, *USM-7, USM-14, USM-20*
- state label, *USM-60*
- state machine names
  - legal characters, *USM-42*
- state names
  - restrictions, *USM-40*
- state variables
  - definition of, *USM-9*
  - Keyword state variable format, *USM-53*
  - legal characters, *USM-53*
  - Positional state variable format, *USM-52*
  - restrictions, *USM-13, USM-14*
  - state variable assignments, (*see* state variable values)
  - state variable names, *USM-52*
    - in SMF Outputs Section, *USM-40*
    - legal characters, *USM-52*
    - restrictions, *USM-40, USM-41*
  - state variable values, *USM-52*
    - legal characters, *USM-52*
- states, *USM-9*
  - power-up state, *USM-20*
  - undefined, *USM-20, USM-41*
- States subsection, (*see* State Machine File)

stepper motor controller, (*see* EXAMPLE2)  
subnodes, (*see* nodes)  
suggestions  
    sending via Fido, *RE-15*  
SYMBOLS command, (*see* FSIM commands)  
synchronous clocks, (*see* Clock signal)

## T

T\_TAB:, (*see* State Machine File: Truth Table Section)  
TABLE keyword, (*see* Functional Simulator: Vector File)  
tabs, (*see* white space)  
TBL file, (*see* Functional Simulator: Watch File)  
Telink, *RE-10*  
text editor  
    invoking from A+PLUS, *R3-3*, *U3-3*  
tildes in pin names, *UBE-19*, *USM-49*, *R2-7*  
Title Block, *R1-3*  
    usage with MACRO and ADLIB, *R1-4*  
TOIF, *UBE-20*, *R1-43*  
TONF, *R1-44*  
TOTF, *R1-45*  
Transitions subsection, (*see* State Machine File)  
transitions  
    order of evaluation, *USM-14*, *USM-41*  
    transition syntax  
        Case statement transitions, *USM-55*  
        conditional transitions, *USM-54*  
        unconditional transitions, *USM-54*  
    use of auxiliary variables, *USM-41*  
Translator, (*see* Altera Design Processor)  
tri-state buffer, *UFS-48*  
Truth Table Section, (*see* State Machine File)  
TTL MacroFunctions, *U1-5*  
Turbo-Bit, *R1-4*  
Turbo-Bit  
    default value, *UBE-18*, *R2-6*, *RB-3*  
    restrictions, *UBE-35*, *USM-41*, *R1-4*, *R2-6*  
    toggling with LogicMap II, *UBE-35*, *USM-41*, *R2-6*

## U

unconditional outputs, (*see* outputs: state machine outputs)  
unconditional transitions, (*see* transitions)  
unconnected pins, *R4-3*



- undefined logic levels, *UFS-12*
  - propagation of, *UFS-48*
- Unused Resources Section, (*see* Utilization Report)
- unused pins
  - in Utilization Report, *R4-3*
  - reserving for future use, *R4-3*
- updates
  - software, *U2-2, RE-4*
- uploading files, (*see* files)
- UTILITIES**, (*see* LogiCaps)
- Utilization Report, *U1-12, R3-10*
  - clock groups, *R4-5*
  - exclamation points, *R4-4*
  - feedback groups, *R4-5*
  - latch numbers, *R4-5, R4-6*
  - macrocell numbers, *R4-4*
  - pins
    - assignments, *R4-1*
    - reserved, *R4-3*
    - unconnected, *R4-3*
    - unused, *R4-3*
  - product terms, *R4-5*
  - sample RPT files, *R4-8–25*
- Sections
  - Buried Registers, *R4-4*
  - Header, *R4-3*
  - Latches, *R4-4*
  - Macrocell Interconnection Cross Reference, *R4-6*
  - Outputs, *R4-4*
  - Part Utilization, *R4-5*
  - Unused Resources, *R4-4*

## V

- VCC**, *R1-17*
- VEC file, (*see* Functional Simulator: Vector File)
- VECTOR** command, (*see* FSIM commands)
- vectors, (*see* Functional Simulator)
- Virtual Logic Analyzer
  - buses
    - combining, *ULA-15*
    - creating, *ULA-15, ULA-23*
    - display of bus value, *ULA-15*
    - expanding, *ULA-15*
  - creating node list, *UFS-75*

## Virtual Logic Analyzer – *continued*

### cursors

- locking together, **ULA-20**
- moving, **ULA-19**
- node cursor, **ULA-8**
- toggle active and inactive, **ULA-19**
- waveform cursor, **ULA-9**

### data buffer, **ULA-2–3**

- searching, **ULA-11**

### exiting, **ULA-24**

### general description, **ULA-2–3**

### help, **ULA-18**

### highlighting breakpoints, **ULA-22**

### invoking, **UFS-84, ULA-4**

### nodes

- adding, **ULA-14, ULA-22**
- combining into buses, **ULA-15, ULA-23**
- deleting, **ULA-14, ULA-22**
- expanding buses into nodes, **ULA-15, ULA-23**
- flashing, **ULA-22**
- moving, **ULA-14, ULA-23**
- selecting, **ULA-22**

### subcommands

- description, **ULA-17–24**

### use in batch mode, **ULA-4**

### vector clock cycle, **ULA-9**

### waveform discontinuities, **ULA-7**

### windows

- description, **ULA-5–6**
- flashing indicator, **ULA-6**
- panning, **ULA-8**
- splitting, **ULA-11**
- zooming, **ULA-7**

VLA, (*see* Virtual Logic Analyzer)

## W

### Warning messages

A+PLUS, **RM-41**

Functional Simulator, **U(FS)M-12**

State Machine Converter, **U(SM)M-11**

Warranty, (*see* Extended Software Warranty)

WATCH command, (*see* FSIM commands)

**WAV** file, (see Functional Simulator: Plot File)  
white space, **UBE-35**  
    in ADFs, **R2-3**  
    in SMFs, **R2-46**  
Write Enable signal, **UBE-33**  
    inputs to bus primitives, **UBE-21**  
Write Strobe signal  
    active low logic, **UBE-31**  
    effects on vector processing in FSIM, **UFS-42**  
    inputs to bus primitives, **UBE-21**

## **X**

Xmodem, **RE-10**  
XNOR, **R1-18**  
XOR, **R1-19**





**ALTERA**

**ALTERA CORPORATION  
3525 MONROE STREET, SANTA CLARA, CA 95051  
(408) 984-2800**