

**PLDshell Plus[™] / PLDasm[™]
User's Guide V5.0**

Copyright © 1994-1996, Altera Corporation. All rights reserved

The installation program used to install PLDshell Plus, *INSTALL*, is based on licensed software provided by Knowledge Dynamics Corp, Highway Contract 4 Box 185-H, Canyon Lake, Texas 78133-3508 (USA), 1-512-964-3994. *INSTALL* is Copyright © 1987-1992 by Knowledge Dynamics Corp which reserves all copyright protection worldwide. *INSTALL* is provided to you for the exclusive purpose of installing PLDshell Plus. Altera has made modifications to the software as provided by Knowledge Dynamics Corp, and thus the performance and behavior of the *INSTALL* program shipped with PLDshell Plus may not represent the performance and behavior of *INSTALL* as shipped by Knowledge Dynamics Corp. Altera is exclusively responsible for the support of PLDshell Plus, including support during the installation phase. In no event will Knowledge Dynamics Corp be able to provide any technical support for PLDshell Plus.

PLDshell Plus contains portions of Vermont Views™ software Copyright 1988, 1991 Vermont Views Creative Software. All rights reserved. Vermont Views is a trademark of Vermont Creative Software.

Altera is a registered trademark of Altera Corporation. The following are trademarks of Altera Corporation: PLDshell, PLDshell Plus, PLDasm, PENGN, Turbo Bit, FLEXlogic, Classic, EPX8160, EPX780, EPX740, EP1810, EP1800, EP910, EP900, EP610, EP600, EP330, EP320, EP310, EP324, EP312, EP220, EP22V10, EP22V10E. Altera acknowledges the trademarks of other organizations for their respective products or services mentioned in this document, specifically: PAL and PALASM are registered trademarks of Advanced Micro Devices, Inc. Hercules is a registered trademark of Hercules Computer Technology. Intel is a trademark of Intel Corporation. IBM is a registered trademark and PS/2 is a trademark of International Business Machines Corporation. GAL is a registered trademark of Lattice Semiconductor, Inc. MS-DOS is a registered trademark, and Windows and Windows NT are trademarks of Microsoft Corporation. ESPRESSO mv-II is copyrighted by the University of California Regents, Berkeley. Altera products marketed under trademarks are protected under numerous U.S. and foreign patents and pending applications, maskwork rights, and copyrights. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera Corporation. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.

U.S. and European patents pending

Getting Started — Installation

This Chapter covers the following installation information:

- System Requirements
- DOS Installation
- Installation Notes
- Configuration Notes

System Requirements

PLDshell Plus is designed to work on MS-DOS systems that are configured as follows:

- IBM-compatible 80386 or higher with 640K bytes of RAM and at least 2M bytes of XMS (extended) memory (4M bytes recommended).

NOTE:

You may need to increase system RAM beyond 4M bytes when compiling large device files, to avoid out of memory errors.

- MS-DOS V5.0 or later.
- High-density (1.44 Megabyte) 3.5-inch diskette drive.
- Hard disk with approximately 5 Megabytes of space (4 Megabytes for installation; up to 1 Megabyte for working files while running).
- A VGA monitor/video card is required to view simulation vectors as waveforms.

Using Extended (XMS) Memory

Using PLDshell Plus with Extended (XMS) Memory requires an extended memory manager such as HIMEM.SYS which is included in Windows 3.1 or DOS 5.0 and above. (Refer to your Windows 3.1 or MS-DOS documentation for complete information on these programs.) The XMS driver should be included in your CONFIG.SYS file as follows:

```
DEV=C:\DOS\HIMEM.SYS
DEV=C:\DOS\EMM386.EXE NOEMS
```

PLDshell Plus EMS Usage Notes

If EMS memory is installed, PLDshell Plus will attempt to swap its memory image into EMS memory. The memory requirements for PLDshell Plus vary from 300 Kbytes to 400K bytes. If PLDshell is unable to swap to EMS memory, it will attempt to swap to disk. The swap disk space must be of the same size and the disk must be writable. A RAM disk can be used instead of either EMS memory or a magnetic disk.

The type of swap to be used (EMS, disk, or none) is determined by commands in the file `PLDSHELL.CFG`:

SWAP — determines the type of swap area, if any: `TRYEMS` (default) `NOEMS`, or `NOSWAP`

SWAPPATH — determines the path to the drive used for the swap, if any

SWAP has the following options:

SWAP TRYEMS — tries EMS; if no EMS, swaps to disk

SWAP NOEMS — prevents swapping into EMS

SWAP NOSWAP — prevents swapping

The options for `SWAPPATH` are specified drives/directories or the current working directory. For example:

```
SWAPPATH c:\tmp .
```

tells PLDshell to attempt to swap into the `TMP` directory of drive `C`. If that fails, then PLDshell will use the current working directory (default). The `SWAPPATH` is specified in a manner similar to the `IPLSPATH` variable.

SWAP and SWAPPATH have the following defaults:

```
SWAP TRYEMS
SWAPPATH .
```

DOS Installation

Install PLDshell Plus as follows:

1. Insert diskette #1 into the A: drive of your system. Type:

```
A:INSTALL<Enter>
```

2. As the INSTALL program begins executing, it displays system resources and prompts you for information such as the target drive, target directory, etc. You can press <Enter> to accept each default or can specify different names/values.
3. INSTALL prompts you for the name of your text editor. You can type the name and press <Enter> or just press <Enter> to accept the default. EDIT (DOS editor) is the default editor name.
4. INSTALL displays messages as it expands archived files and copies those files to the target drive and directory. INSTALL prompts you to insert the next disk.
5. INSTALL prompts you before changing your AUTOEXEC.BAT and CONFIG.SYS files. If you prefer, you can skip these steps and make these changes via your text editor. The following changes should be made to the CONFIG.SYS file:

```
FILES=20
BUFFERS=15
DEVICE=C:\PLDSHELL\STDWOUT.SYS
```

The number for FILES and BUFFERS can be set higher, but should not be set lower. This example assumes that C:\PLDSHELL is the install drive and directory. Include your actual install drive and directory.

The driver, STDWOUT.SYS, provides a PLDshell video driver.

6. The following line must appear in your AUTOEXEC.BAT file:

```
SET PLDSHELL=C:\PLDSHELL\PLDSHELL.CFG
```

This example assumes that C:\PLDSHELL is the install drive and directory. Include your actual install drive and directory. Do *not* include spaces before or after the equal sign.

7. Reboot your system to load the modified AUTOEXEC.BAT and CONFIG.SYS files. You can now run PLDshell Plus/PLDasm.

Please refer to the following section if you encounter any problems with installing PLDshell Plus.

Installation Notes

The following notes provide additional installation information:

1. Be sure to refer to the READ.ME file(s) on the PLDshell Plus diskettes for late-breaking installation and operation information.
2. The program HIMEM.SYS is not compatible with VDISK.SYS. If both programs are installed, the result will be no extended memory available for other application programs. Instead, use the RAMDRIVE.SYS program.

NOTE:

PLDshell Plus may not be compatible with some versions of disk caching programs. If you experience a problem with installing PLDshell Plus, disable the disk cache program before installing PLDshell Plus.

3. After running the installation and rebooting your machine, type the SET command at the DOS prompt to check that the following environment variables are set correctly:
 - The PLDshell environment variable must be set as:

```
PLDSHELL=C:\PLDSHELL\PLDSHELL.CFG
```

- The path must contain the PLDshell installation directory. Make sure the PATH command includes PLDSHELL along with other directories. It will look something like this:

```
PATH=C:\DOS;C:\PLDSHELL;C:;. . .
```

If these statements are missing or incomplete, increase the environment space by including the SHELL command in your CONFIG.SYS file as follows:

```
SHELL=C:\COMMAND.COM /P /E:2048
```

where 2048 designates memory to be reserved for environment space. The number 2048 is only an example; a recommended number is 128 bytes higher than the current number. Do not include spaces before or after the equal sign. Reboot your system to load the modified configuration file.

4. DOS recognizes only the first 127 bytes of a PATH command in an AUTOEXEC.BAT file. Please note this DOS limitation. If you are encountering search path problems when running PLDshell Plus, you should shorten your PATH command by removing some paths. You can still run programs no longer on the path by including them in the Run menu. With DOS 5.0 you can include APPEND statements in your AUTOEXEC.BAT file to search additional paths for files. Refer to your DOS User's Guide for details.
5. When installing PLDshell Plus on network systems, create local configuration files for each user. Make the PLDshell Plus install directory on the network drive read-only to avoid accidental modifications to executable and library files. A local AUTOEXEC.BAT file, for example, should contain the following line pointing to the local configuration file (C: is the local drive):

```
SET PLDSHELL=C:\NETUSER\PLDSHELL.CFG
```

Two variables in the local PLDSHELL.CFG should point to the executable and library files residing on the network drive, as shown below (F: is the network drive):

```
IPLSPATH F:\PLDSHELL\
INCLUDE F:\PLDSHELL\
```

6. It is recommended that PLDshell Plus be installed over any previous versions of PLDshell Plus you have installed. If you do install PLDshell Plus on a system where a previous version is already present (i.e., in a different directory), make sure you create batch files to set/reset the PLDSHELL environment variable to point to the correct configuration file. You also need to change the order of both directories in the search path to ensure proper operation. This can also be included in the batch files.
7. Note that the modifications to the CONFIG.SYS and AUTOEXEC.BAT files are made as additions to the end of the existing files. If the boot procedure already turns control of the system over to an application program via the CONFIG.SYS or AUTOEXEC.BAT file, these additions may never be executed while booting your system. If this is the case, PLDshell Plus will not run. To correct this problem, edit your files to change the order of the DOS commands related to PLDshell Plus variables.

Configuration Notes — Windows 3.x

1. PLDSHELL and PLDASM may be run in a Windows-hosted DOS shell, however, technical support for problems experienced while running from within Windows is not provided, and the product has not been tested extensively under the Windows environment.

The known problems running PLDSHELL/PLDASM under Windows are listed below:

- a. A System halt and/or corrupted display may occur when operating the Waveform Viewer under Windows.
 - b. If the AUTOEXEC.BAT boots Windows, it is possible the PLDSHELL environment variable may not be set properly. Make sure the command to boot Windows is the last line of the AUTOEXEC.BAT after the PLDSHELL installation is completed.
2. If you want to run PLDshell Plus under Windows, create a PIF that provides a minimum of 2M bytes of extended memory. Refer to your Windows documentation for instructions
 - a. Select **File/New** in the Program Manager dialog box.

- b. Enter a descriptive name in the Description field.
- c. Enter the command line (including the path name) that will run PLDshell Plus, in this case “C:\PLDSHELL\PLDSHELL.EXE”.
- d. If you wish, you can select an icon from the icons provided by Windows to use to run PLDshell Plus from the Program Manager.

The mouse cursor should function in PLDshell in the DOS shell full-screen mode for both versions of Windows.

Using Windows 3.1

If you are running Windows 3.1 in the 386 enhanced mode, you have the option of running a DOS application in a window. This is done by pressing the <Alt-Enter> keys. The same combination of keys will return the DOS application to the full-screen mode.

The mouse cursor may function in a DOS window under Windows 3.1 if you have the appropriate video and mouse driver. It should function in the full-screen mode.

Chapter 1 — Introduction

PLDshell Plus provides an easy-to-use menu system for logic device design that allows you to invoke Altera's PLDasm compiler, or your existing PLD logic compilers.

The PLDshell Plus Main Menu (shown in Figure 1-1) is arranged to follow the typical design flow: Edit, Compile/Sim, and View.

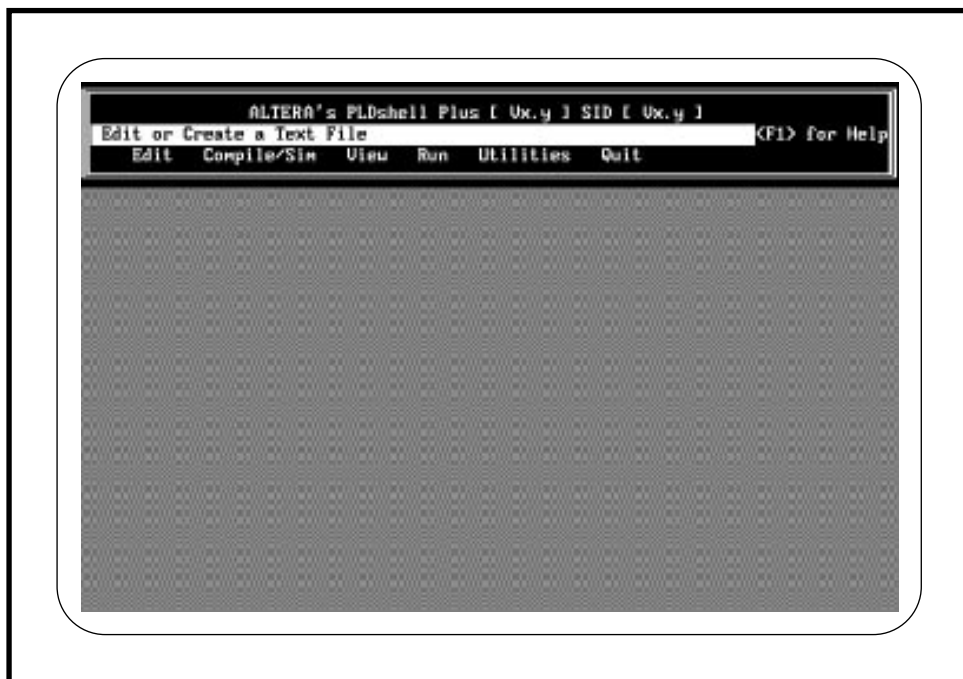


Figure 1-1 PLDshell Plus Main Menu

PLDasm is a logic compiler, estimator, and functional simulator that runs under PLDshell Plus. PLDasm compiles PALASM2-compatible source files to produce JEDEC files for a variety of Altera devices, including the FLEXlogic device family. This software allows you to use a familiar design language to evaluate the architecture of Altera devices and to implement new designs.

| PLDshell Plus MENU OPTIONS | | | | |
|---|---|---|--|--|
| EDIT | COMPILE/ SIM | VIEW | RUN | UTILITIES |
| Create and Revise Source Files | PLDasm Compiler, Estimator, and Simulator | View Error, Report, Waveform, and Other Files | Run Other Programs (e.g., APT, PENGNG, and JED2JTAG) | Disassemble, Convert, Translate, Merge, and Other Utilities |

Figure 1-2 PLDshell Plus Main Menu Options/Functions

As shown in Figure 1-2, PLDshell Plus menu options allow you to:

- Edit device source files using your preferred text editor.
- Compile, estimate, and simulate source files to produce JEDEC programming files for Altera devices.
- View error message, report, and waveform files for designs.
- Run other design tools or programs, such as the APT software for programming Classic devices, and the PENGNG and JED2JTAG programs for programming/configuring FLEXlogic devices.
- Run utilities that allow you to disassemble JEDEC files for supported devices into PLDasm source files for Altera devices (JEDEC file to PLDasm file), or to perform a full conversion (PAL/GAL JEDEC file to Altera device JEDEC file).
- Merge from two to sixteen PLDasm source files into a single PLDasm source file that can be compiled or estimated.

PLDasm Design Compilation Overview

PLDasm compiles PALASM2-compatible source files to produce JEDEC files for Altera devices. Figure 1-3 shows how the PLDasm compiler fits into the overall design flow.

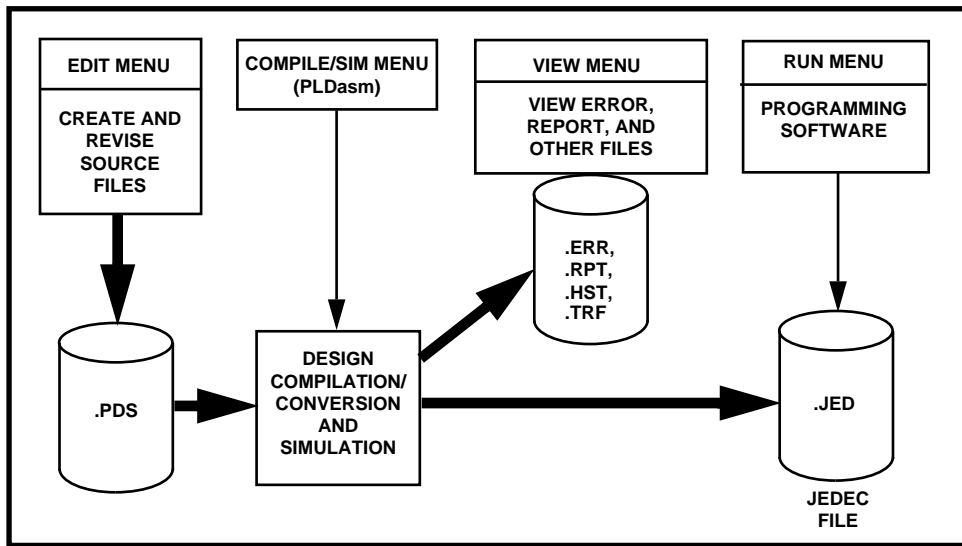


Figure 1-3 PLDshell Design Compilation Overview

The typical design process is to create the source file, compile/simulate the design to create a JEDEC file, and program devices. Error, report, and waveform files are viewed throughout the cycle. The edit/compile/simulate/view process is repeated until a design is working as desired. Devices are programmed at the end of the cycle. PLDasm offers the following features:

- Preserves your investment in learning a design language and in developing source files by compiling an industry-standard language.
- Implements designs using Boolean equations, State Machine syntax, or Truth Tables. (Truth Table design is a PLDasm superset feature.)
- Functionally simulates designs.
- Maps designs into device resources and performs extensive logic minimization using the ESPRESSO mv-II logic minimization algorithm.
- Generates JEDEC programming files; these files include programming test vectors based on simulation output.

Table 1-1 lists all Altera devices supported by PLDshell Plus/PLDasm, as well as the device names that are recognized in .PDS files (including equivalent Intel names).

Table 1-1 Altera Devices Supported by PLDshell Plus/PLDasm

| Family | Altera Device | Package | Device Names (Accepted in .PDS Files) |
|------------------|---------------|-----------------|---|
| FLEXlogic Family | EPX8160 | QFP | EPX8160QC208, FX8160_208 |
| | EPX780 | QFP (132 pin) | EPX780QC132, FX780_132 |
| | | J-Lead (84 pin) | EPX780LC84, EPX880LC84, FX780_84 |
| | EPX740 | J-Lead (68 pin) | EPX740LC68, FX740_68 |
| | | J-Lead (44 pin) | EPX740LC44, FX740_44 |
| Classic Family | EP1810/EP1800 | J-Lead | EP1810LC, EP1810JC, EP1800ILC, N5C180 |
| | | PGA | EP1810GC, EP1800IGC, G5C180 |
| | EP910/EP900 | DIP | EP910DC, EP910PC, EP910IDC, EP900IDC, EP900IPC, PLD910, 5C090, 85C090 |
| | | J-Lead | EP910JC, EP910LC, EP910ILC, EP900ILC, PLD910N, N5C090, N85C090 |
| | EP610/EP600 | DIP | EP610DC, EP610PC, EP610IDC, EP610IPC, EP600IDC, EP600IPC, PLD610, 5C060, 85C060 |
| | | SOIC | EP610SC |
| | | J-Lead | EP610LC, EP610ILC, EP600ILC, PLD610N, N5C060, N85C060 |
| | EP324 | DIP | EP324DC, EP324PC, 5AC324 |
| | | J-Lead | EP324LC, N5AC324 |
| | EP312 | DIP | EP312DC, EP312PC, 5AC312 |
| | | J-Lead | EP312LC, N5AC312 |
| | EP330/EP320 | DIP | EP330PC, EP320IDC, EP320IPC, 5C032 |
| | | SOIC | EP330SC |
| | | J-Lead | EP330LC |
| | EP310 | DIP | EP310IDC, 5C031 |
| | EP224 | DIP | EP224DC, EP224PC, 85C224 |
| | | J-Lead | EP224LC, N85C224 |

Table 1-1 Altera Devices Supported by PLDshell Plus/PLDasm (Continued)

| Family | Altera Device | Package | Device Names (Accepted in .PDS Files) |
|-------------------------------|----------------------|----------------|---|
| Classic Family (continued) | EP220 | DIP | EP220DC, EP220PC, 85C220 |
| | | J-Lead | EP220LC, N85C220 |
| | EP22V10 | DIP | EP22V10PC, PLD22V10 |
| | | J-Lead | EP22V10LC, PLD22V10N |
| | EP22V10E | DIP | EP22V10EPC, 85C22V10 |
| | | J-Lead | EP22V10ELC, N85C22V10 |

You can also compile .PDS files for common PAL/GAL devices using PLDasm. PAL/GAL designs are transparently converted (i.e., retargeted) into JEDEC files for the appropriate Altera device. Table 1-2 lists PAL/GAL devices that can be converted by PLDshell, and the Altera equivalent device to which each is converted..

Table 1-2 Other Supported PLDs, Packages, and Device Names

| Other PLDs, PALs, and GALs | Package | Device Names (Accepted in .PDS Files) | Type of JEDEC File Created |
|----------------------------|---------|--|----------------------------|
| 16L8 | DIP | 16L8 | EP220PC |
| | PLCC | 16L8NL | EP220LC |
| 16R4 | DIP | 16R4 | EP220PC |
| | PLCC | 16R4NL | EP220LC |
| 16R6 | DIP | 16R6 | EP220PC |
| | PLCC | 16R6NL | EP220LC |
| 16R8 | DIP | 16R8 | EP220PC |
| | PLCC | 16R8NL | EP220LC |
| 16V8 | DIP | 16V8 | EP220PC |
| | PLCC | 16V8NL | EP220LC |
| 20L8 | DIP | 20L8 | EP224PC |
| | PLCC | 20L8FN | EP224LC |
| 20R4 | DIP | 20R4 | EP224DC |
| | PLCC | 20R4FN | EP224LC |
| 20R6 | DIP | 20R6 | EP224PC |
| | PLCC | 20R6FN | EP224LC |
| 20R8 | DIP | 20R8 | EP224PC |
| | PLCC | 20R8FN | EP224LC |
| 20V8 | DIP | 20V8 | EP224PC |
| | PLCC | 20V8FN | EP224LC |
| 22V10 | DIP | 22V10 | EP22V10PC |
| | PLCC | 22V10FN | EP22V10LC |
| 22VP10 | DIP | 22V10 | EP22V10EPC |
| | PLCC | 22V10FN | EP22V10ELC |

Note: For PAL/GAL names, PLDshell Plus/PLDasm makes an attempt to recognize the most common manufacturers' device names. If the compiler does not recognize the name given in the .PDS file, try changing the device name in the file so that it matches one of the names listed in Table 1-2.

JEDEC Disassembly Overview

Under the Utilities Menu, PLDshell Plus provides the ability to disassemble existing JEDEC files for the supported Altera devices into PLDasm source files. Disassembly is supported for all Altera Classic devices, as listed in Table 1-1. You can also disassemble JEDEC files for the common 20- and 24-pin PALs and GALs listed in Table 1-2, as shown Figure 1-4 below.

JEDEC disassembly allows you to reconstruct source files for existing designs where the original source files have been lost, or to generate source files from existing designs to be modified for new designs. Note that the source file output during the disassembly process is for the respective Altera device.

JEDEC disassembly is available via the Utilities—Disassemble menu selection.

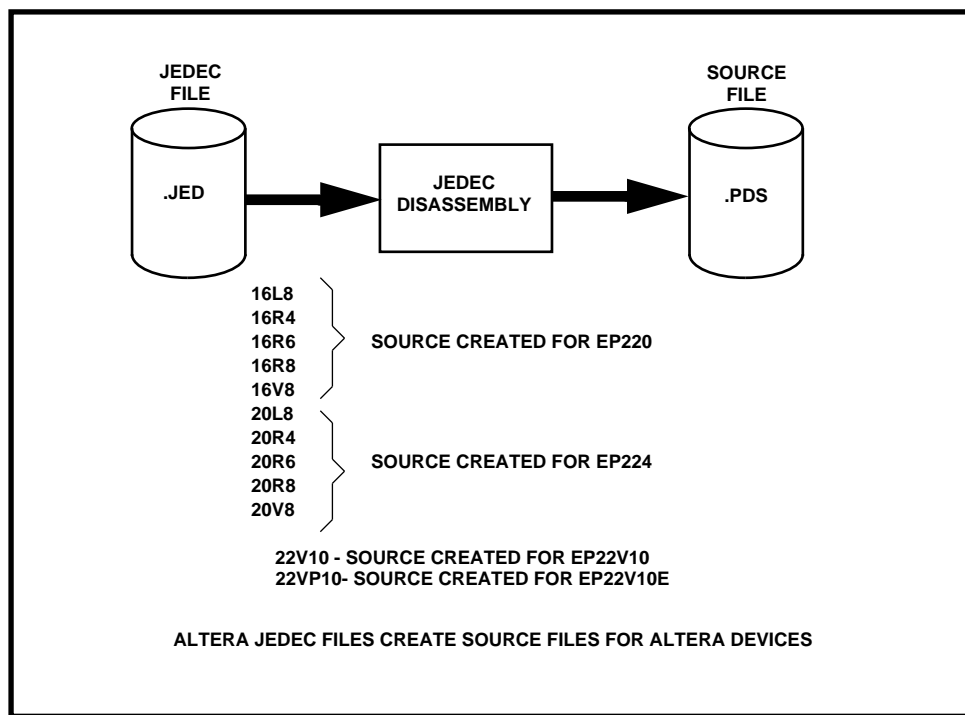


Figure 1-4 JEDEC Disassembly to PLDasm Source File

JEDEC Conversion Overview

Under the Utilities Menu, PLDshell Plus provides the ability to convert existing JEDEC files for common PALs/GALs into JEDEC files for Altera devices. A PLDasm source file is automatically generated during the conversion process (see Figure 1-5). Conversion guarantees that the target Altera device is functionally the same as the original design. Table 1-2, Other Supported PLDs, Packages, and Device Names, lists the devices supported by the conversion.

JEDEC conversion is available via the Utilities—Convert JEDEC menu selection.

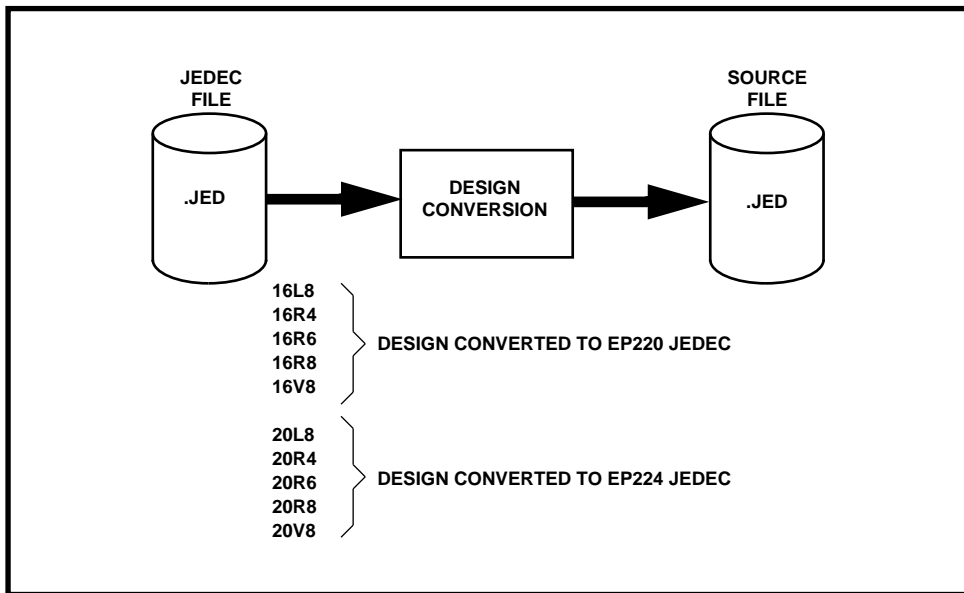


Figure 1-5 JEDEC Conversion

.ADF/.SMF Translation Overview

PLDshell Plus has the capability to translate .ADF/.SMF files into .PDS files that can be compiled by PLDasm. Figure 1-6, .ADF/.SMF Translation into .PDS File, illustrates the process. The .SMF files are processed through an intermediate program called ISTATE (Intermediate State Machine Translator) into .ADF files. The .ADF files are then translated into .PDS files.

Translation is available via the Utilities—Translate menu selection.

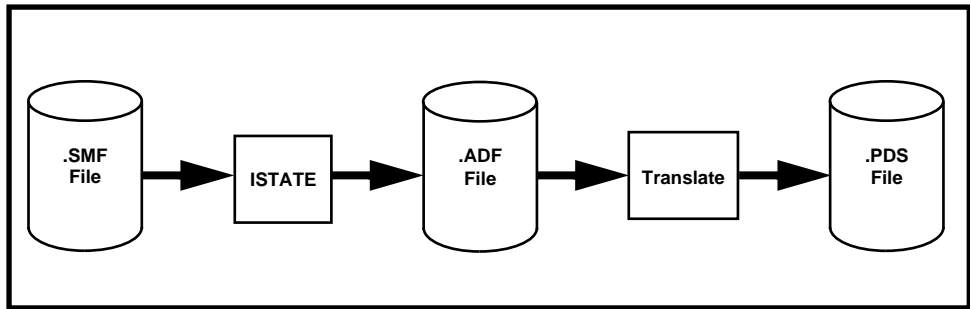


Figure 1-6 .ADF/.SMF Translation into .PDS File

Merge Overview

PLDshell Plus allows you to merge from two to sixteen .PDS files into a single .PDS source file. The merged .PDS file can then be compiled, simulated and/or estimated. Figure 1-7 illustrates the process.

The Merge utility is available via the Utilities—Merge menu selection.

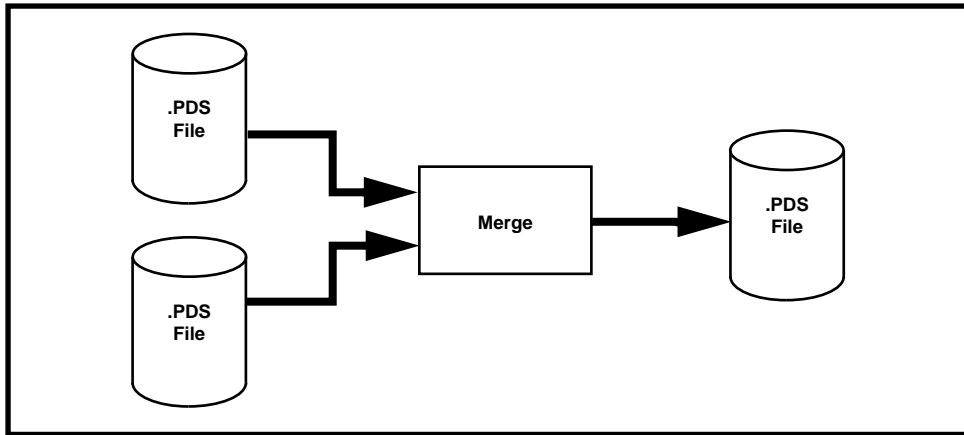


Figure 1-7 File Merge Utility

Device Programming

PLDshell Plus includes the PENGN and JED2JTAG programs for in-circuit programming and configuration of FLEXlogic devices with the Altera FLEXlogic Prototyping Cable Kit (available separately). For more information on the FLEXlogic Prototyping Cable, PENGN, and JED2JTAG, see Appendix F, FLEXlogic Prototyping Cable.

PLDshell Plus also includes the APT (Advanced Programming Tool) software for programming Altera Classic devices with the Intel iUP-PC (Universal Programmer Personal Computer) or iUP-200A/201A Universal Programmer, each of which relies on the Intel iUP-GUPI Module Base and GUPI programming adapters. APT can be invoked from the Run Menu in the current version of PLDshell Plus. (The APT-compatible Intel programming hardware is no longer available. For information on using APT with Intel programming hardware, refer to your User's Guide for an earlier version of PLDshell Plus/PLDasm.)

A variety of third-party manufacturers also offer programming support for Altera devices. For additional information on Altera and third-party programming support, refer to the current Altera Data Book.

Additional PLD Design Tools

Other design tools are available to speed your design efforts with PLDshell Plus/PLDasm and/or allow you to squeeze more functionality into a target device. PLDshell Plus/PLDasm can interface with a variety of leading third-party CAE tools. For additional information, contact Altera Applications at:

Altera Corporation
Applications Department
2610 Orchard Parkway
San Jose, CA 95134-2020

Tel: (800) 800-EPLD or (408) 894-7000
Fax: (408) 954-0348

Chapter 2 — Tutorial

This Chapter covers the following information:

- How to invoke PLDshell Plus
- Walking through a sample four-bit counter design

Invoking PLDshell Plus

After the installation is complete, type:

```
PLDSHELL <Enter>
```

NOTE:

If you are planning to use any of the EXAMPLE files, make sure to invoke PLDshell from the installation directory.

Press <Enter> at the initial PLDshell screen that appears and you will see the Main Menu as shown in Figure 2-1. To select a menu item, use the mouse and click, or use the ← and → cursor keys to highlight the item and press <Enter>. Alternatively, you can type the first letter of the menu selection, such as the <E> key for **Edit**.

NOTE:

At any time while running PLDshell Plus, press <F1> to invoke the help feature.

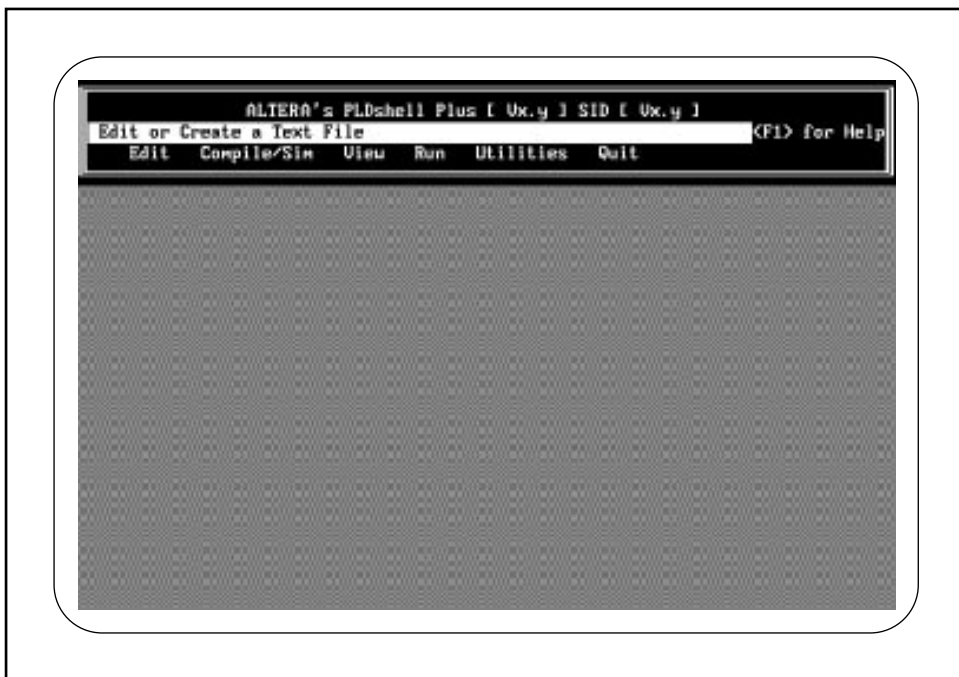


Figure 2-1 PLDshell Main Menu

Sample Design: 4-Bit Counter

This section is a step-by-step tutorial on creating and compiling a design using PLDshell Plus/PLDasm. The example design is a simple 4-bit synchronous counter targeted for an Altera EP224 PLD. You can start at step 1 to create the PLDasm source file as described here and compile it to produce a JEDEC file. You can also use the sample file (4COUNT.PDS) installed with the software; in this case, skip to step 6 and run the compiler.

Step 1: Header and Declarations

Open a source file using the **Edit** option on the Main Menu. Press <F6> and enter the name of the file you wish to create with the extension .PDS then press <Enter>. Pick a base file name that is *not* 4COUNT, or you will overwrite the example.

Create header and declarations sections that contain the following information:

```
Title           4-Bit Counter Sample File
Pattern         pds
Revision        1
Author          Your Name
Company         Your Company
Date            Date

CHIP    4_count    EP224
```

Step 2: Pin Names/Assignments

Enter the pin numbers and pin names for the design as follows (“;” denotes a comment):

```
; pin assignments

PIN    1      CLK    ; clock pin
PIN    2      ENA    ; counter enable
PIN    15     QA     ; LSB
PIN    16     QB
PIN    17     QC
PIN    18     QD     ; MSB
```

Step 3: EQUATIONS Section

The four outputs of the counter (QA-QD) are implemented using Boolean equations. Enter the keyword EQUATIONS, followed by the actual equations as shown below. The name on the left-hand side is the output name. The “:=” characters specify the output as a registered output. The names and symbols on the right-hand side are the equations that implement the circuit. (Type the first equation as shown; the period between the Q and the A is an intentional error. It will be corrected during a later step. A sample file containing this error was shipped with the software; it is called 4ERROR.PDS)

```
; Boolean equations for registers

EQUATIONS

QA := ENA * /Q.A

QB := ENA * QB * /QA
   + ENA * /QB * QA

QC := ENA * QC * /QA
   + ENA * QC * /QB
   + ENA * /QC * QB * QA

QD := ENA * QD * /QA
   + ENA * QD * /QB
   + ENA * QD * /QC
   + ENA * /QD * QC * QB * QA
```

Step 4: Starting the SIMULATION Section

The Simulation section allows you to specify a functional simulation sequence for your designs. Enter the keyword SIMULATION, followed by simulation commands. The first set of commands assigns a vector, specifies signals to be output to a trace file, and sets the counter inputs and registers to known states (ENA = high, CLK = low, register outputs set low). Clock and control signals should always be set before using the preload (PRLDF) command to ensure registers are initialized in the proper state so you do not propagate undefined states. The next set defines a loop in which the counter is clocked four times.

```
SIMULATION

; set up vector and trace
; set to known state, preload registers (all low)

VECTOR COUNT := [ QD QC QB QA ]
TRACE_ON ENA CLK QD QC QB QA
```

```

SETF ENA /CLK
PRLDF /QA /QB /QC /QD; clock set before
                                ; preload command

; count 4 times

FOR X := 0 TO 3 DO
    BEGIN
        CLOCKF CLK
    END

```

Step 5: Completing the SIMULATION Section

The Simulation section is completed in two parts. First, ENA is brought low and the circuit is cycled through four clocks to test the enable signal (with ENA low the circuit resets to zero on the next clock and does not count on subsequent clocks). Then, ENA is brought high again and the counter is clocked ten more times.

```

; disable counting, then try 4 more times

SETF /ENA
    FOR X := 0 TO 3 DO
        BEGIN
            CLOCKF CLK
        END

; enable counting, then count 10 times

SETF ENA
    FOR X := 0 TO 9 DO
        BEGIN
            CLOCKF CLK
        END

TRACE_OFF

; end of simulation

```

Step 6: Running the Compiler

Save the design file, using the appropriate save command for your text editor. Press <Enter> when prompted to return to PLDshell Plus.

Select the **Compile/Sim** option from the PLDshell Plus Main Menu (see Figure 2-2). Move to **Accept** and press <Enter>, or press <F10> to accept the default compile and simulation conditions (the defaults are described in Chapter 5).

The PLDasm compiler runs, but since an (intentional) error exists, the compiler aborts, displaying an error message. (If you use 4COUNT.PDS, this error will not be displayed. You can compile 4ERROR.PDS to generate this error if desired.)

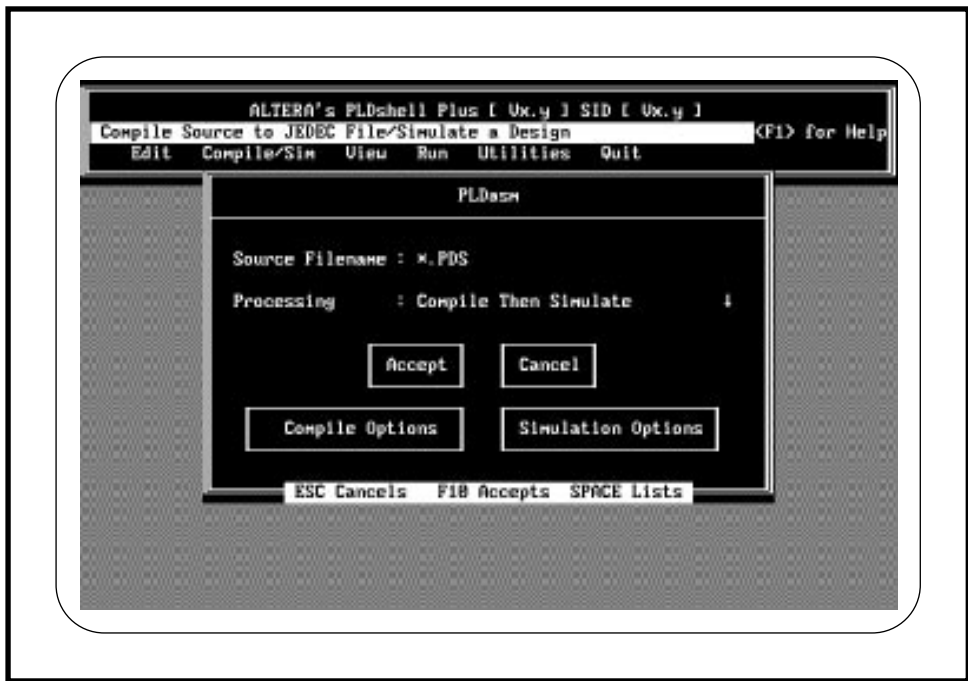


Figure 2-2 PLDshell Plus Compile/Sim Menu

Step 7: Viewing the Error File

Press <Esc> to move back to the Main Menu. Select the **View** option from the PLDshell Plus Main Menu, then select **Error/Log Files**.

Choose the error file that has the same base name as the source file with the .ERR extension and press <Enter>. All error messages are displayed (see Figure 2-3).

```
INFO PARPDS: Parsing file: 4ERROR.PDS.
ERROR E4304-PDSTODDB: Invalid pin name on line 23 at ".".
ERROR E4340-PDSTODDB: Incorrect equal operator on line 25 at "QB".
INFO PARPDS: (0) warning(s), (2) fatal error(s).
```

Figure 2-3 Error File Listing

Using the cursor keys, move the cursor to the first error message and press <F10>. The help message for this error is displayed. The help message also recommends a course of action to correct the error.

Step 8: Revising the Source File

Press <Esc> three times to move back to the Main Menu. Use the **Edit** option to open the source file.

Remove the period from between the Q and the A in the first equation. Save the design and exit the text editor.

Step 9: Recompiling the Design

Press <Esc> to move back to the Main Menu. Select the **Compile/Sim** option and recompile the design to product a JEDEC file. Press <Enter> when done to move to the Main Menu.

Step 10: Viewing the Simulation History File

Select the **View** option from the Main Menu, then select **Vector/Waveform Files**. Choose the simulation history file for the design (same base name as the source file with the .HST extension) and move to **Accept** and press <Enter>, or press <F10>.

View the simulation results (see Figure 2-4). You can move about in the waveform by using the cursor keys. The <+> and <-> keys allow you to zoom in and out.

NOTES:

Press <F1> to see a detailed list of all of the available keystroke options.

Pressing <Tab> from the simulation screen brings up the source file and enables you to toggle between it and the simulation screen.

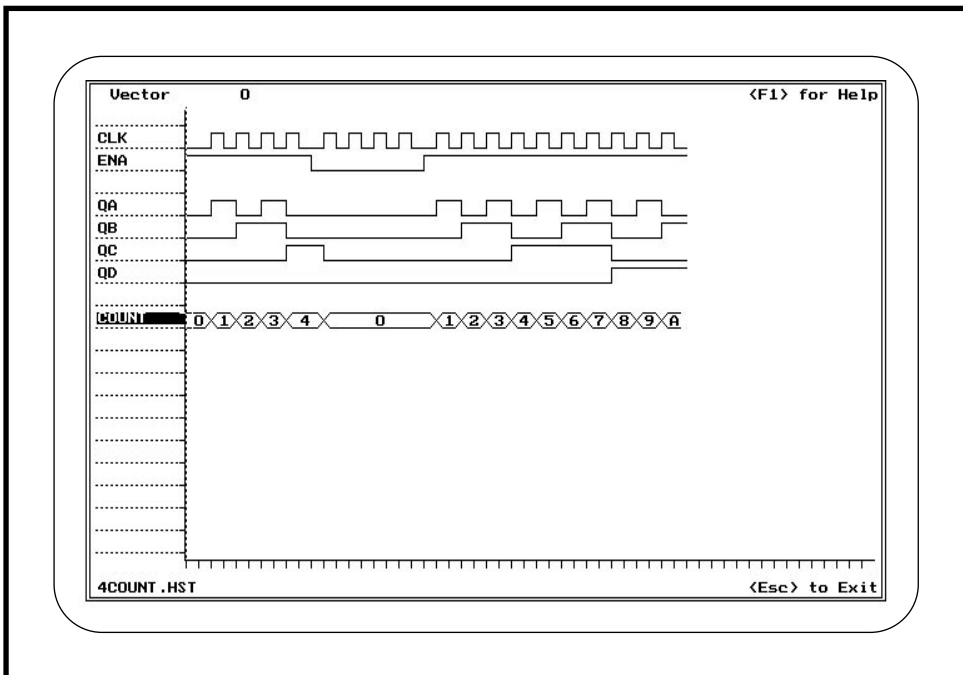


Figure 2-4 PLDshell Plus Viewing Simulation Waveforms

Chapter 3 — Using PLDshell Plus/PLDasm

This Chapter describes the following:

- How to invoke PLDshell Plus
- The contents of and how to use all PLDshell Plus menus

Invoking PLDshell Plus

To invoke PLDshell Plus, from the install directory type:

```
PLDSHELL <Enter>
```

Alternatively, there are three PLDshell program options:

fixkbd — fixes conflicts between PLDshell and some keyboards.

v43 — uses the EGA 43-line mode, requiring EGA monitor and adapter.

v50 — uses the VGA 50-line mode, requiring VGA monitor and adapter.

The following command fixes keyboard conflicts:

```
PLDSHELL FIXKBD <Enter>
```

PLDshell Plus Menu Guidelines

PLDshell Plus features full mouse support as well as access via the keyboard for those who do not have a mouse. The following guidelines will help you use the PLDshell Plus menus and submenus:

- The right mouse button provides access to context-sensitive Help menus. <F1> also displays the same context-sensitive Help information.
- Menu options are selected by:
 - (1) Placing the mouse cursor on a menu item and clicking the left mouse button.
 - (2) Using the ← and → cursor keys to highlight a menu, then pressing <Enter>.
 - (3) Typing the first letter of a menu option.
Submenus use the ↑ and ↓ cursor keys (and <Enter>) or the first letter of the submenu.

- Clicking on a **Cancel** button backs you up one menu level at any time. <Esc> also backs you up one menu level at any time.
- <Enter> executes submenus that do not have options. <Enter> also accepts fields that require information such as file names.
- <F6> clears fields that include text strings (part names, file names, etc.).
- <F9> prints from the View Menu.
- Clicking on an **Accept** button accepts/executes submenus. <F10> also accepts/executes submenus.
- Clicking the left mouse button toggles between options where only two choices are available. Pressing <Space> also toggles between options where only two choices are available.
- Double-clicking the left mouse button displays a list of options where more than two are available. The space bar also displays a list of options where more than two are available. Using the ↑ and ↓ keys will move the cursor through the options. Pressing <Enter> selects a highlighted option.
- Text in option fields, such as file names, can be entered and changed using alphanumeric, backspace/delete, and cursor keys.
- <Home>/<End> keys move to the top or bottom of the file in the View Menu.
- <PgUp>/<PgDn> allow you to quickly scroll the screen in the View Menu.
- Some menus and submenus have **Accept** and **Cancel** buttons as well as other buttons. Select the desired function with the mouse or use the ↑ and ↓ or ← and → cursor keys to highlight these buttons and press <Enter> to activate. Pressing <F10> is the same as **Accept**; pressing <Esc> is the same as **Cancel**. Use of **Accept** for changing options is limited to the current PLDshell Plus/PLDasm session. To set new start-up default values, when available, select the appropriate **Save** button (e.g., **Save Compile Options**), followed by **Accept** or <F10>.

Close Boxes

Figure 3-1 shows an example list window with a **Close** box. List and view windows without **Cancel** or **Accept** buttons have **Close** boxes, which have the same function as pressing the <Esc> key.

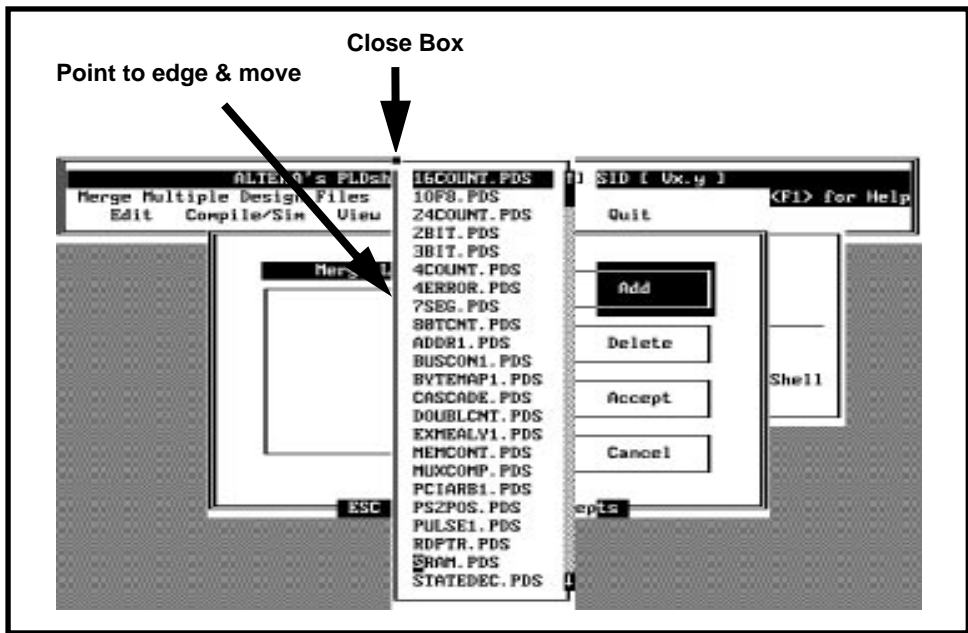


Figure 3-1 Window Close Boxes

Resizable/Moveable View Windows

Figure 3-2 shows a view window that can be resized or moved with the mouse. Place the mouse cursor on the sizing button at the bottom right corner of the window and use click and drag to resize the window. To move a view or list window, place the mouse cursor on any window border, except a scroll bar (if any), and click and drag the window.

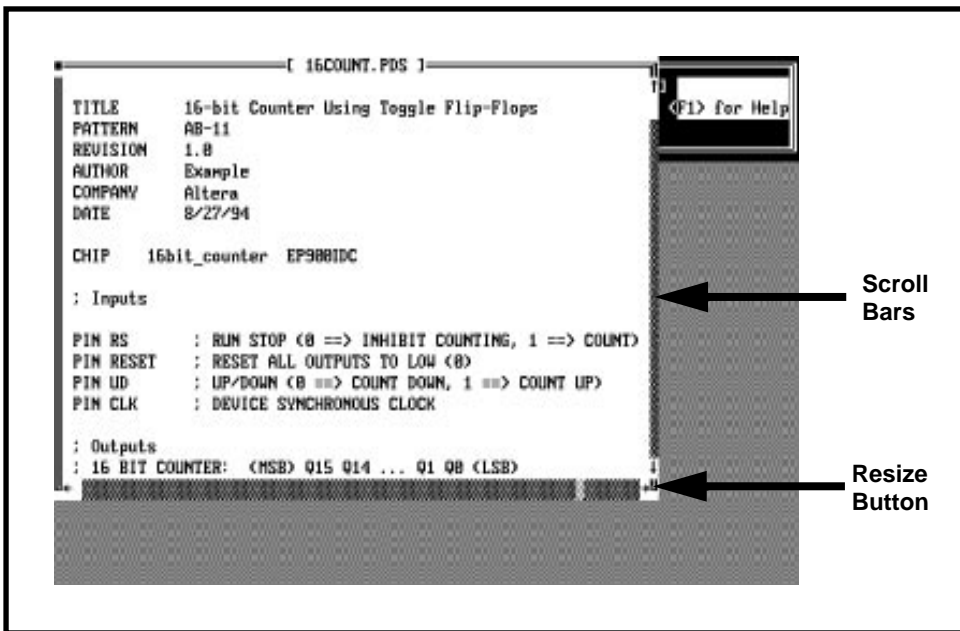


Figure 3-2 Resizable View Windows

NOTE:

Any corner without a close box can be resized.

Scroll Bars

View windows have scroll bars at the right and bottom borders (Figure 3-2). To scroll through a file, place the mouse cursor in the grayed areas of the scroll bar and click the left mouse button. You can also click on the arrows at each end of a scroll bar.

Help

Figure 3-3 shows an example of the Help/Status screen. This screen provides a first level of help.

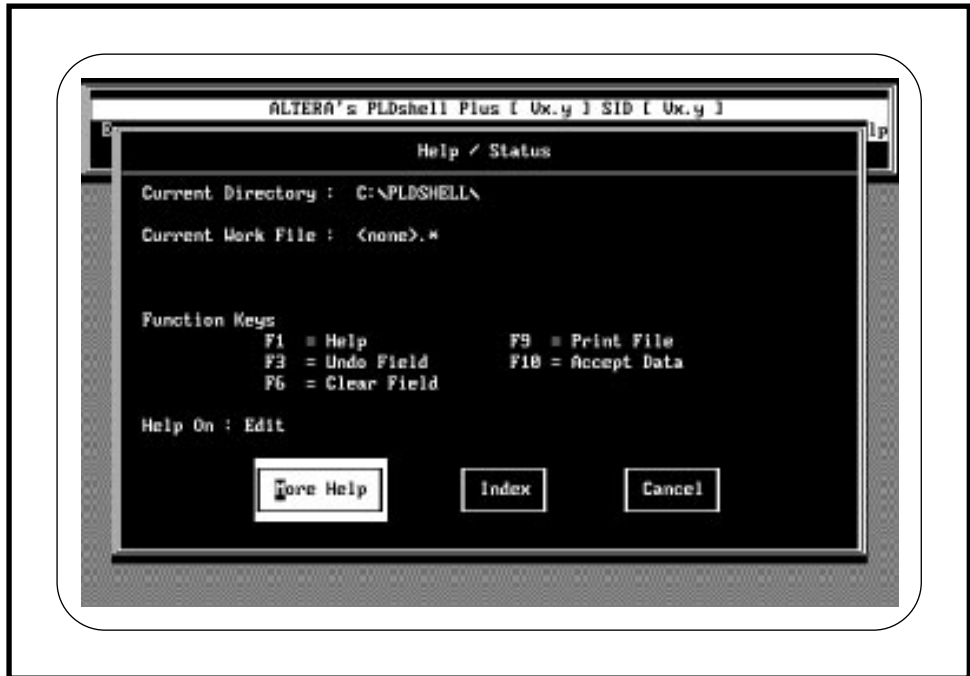


Figure 3-3 Example Help/Status Screen

The Help/Status window displays:

- The current working directory
- The current work file and up to eight associated files (e.g., 7SEG.PDS, 7SEG.ERR, 7SEG.RPT, 7SEG.JED)
- A function key summary
- The current Help topic

There are three buttons:

More Help — Displays more Help information on the current topic.

Index — Displays a logically ordered list of Help topics (see Figure 3-3).

Cancel — Cancels the help facility and returns to the current menu or submenu.



Figure 3-4 Help Index Screen

You can also access on-line help at anytime by pressing <F1>. Additional help information is provided for error messages: when viewing an error file (.ERR), position the cursor on the error number and press <F10> to access additional help.

PLDshell Plus Main Menu

PLDshell Plus supports the following functions from the Main Menu (see Figure 3-5).

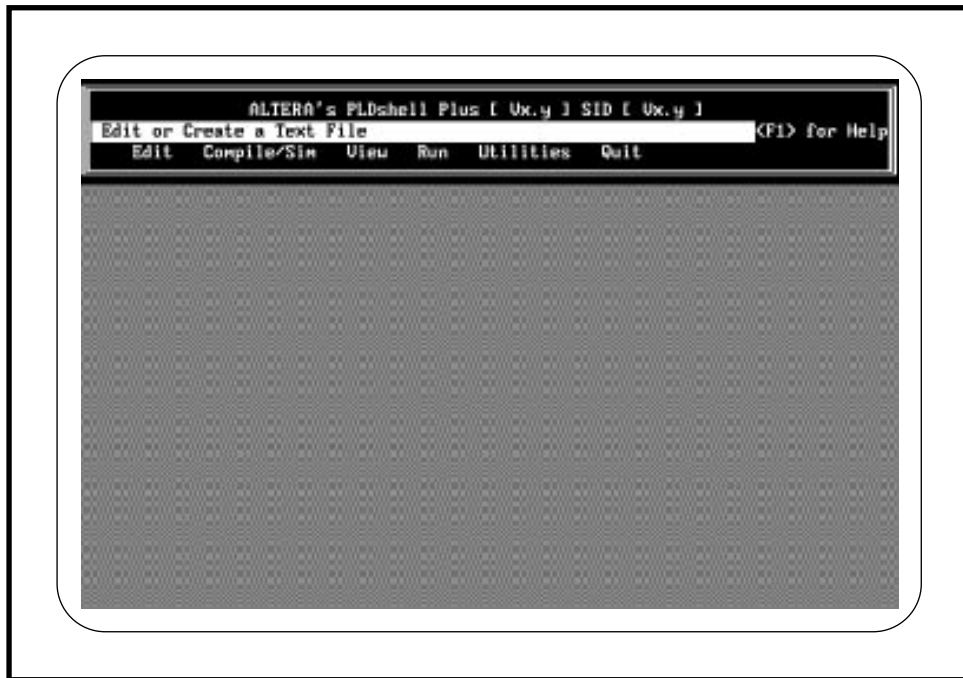


Figure 3-5 PLDshell Plus Main Menu

Edit — Edit source files or any text file using your preferred ASCII text editor.

Compile/Sim — Compile, estimate, and simulate PLDasm source files to create JEDEC files for target Altera devices. You can define compile/simulate options.

View — View PLD source, error, report, or simulation files to quickly locate design or fitting problems.

When viewing an error file, you can move to the desired message and press <F10> to display on-line error message help information. Error message descriptions are also provided in Appendix D of this manual.

Simulation results can be viewed in a table or as waveforms. (Waveform display is supported on Hercules, EGA, and VGA monitors only.)

Run — Run up to 24 user-defined programs including other PLD development tools. The menu is user-defined, with each menu option including default command line options and working directories.

Utilities — Provides several utility functions, including:

Disassembly of JEDEC files to PLDasm files.

Conversion of non-Altera PLD JEDEC files to Altera PLD JEDEC files.

Translation of .ADF/.SMF files created for A+PLUS to PLDasm files.

Merging two to sixteen .PDS files into a single .PDS file for compiling, estimating, or simulation.

Listing the current directory.

Changing the current directory.

Invoking a DOS shell. Note that PLDshell Plus creates a swap file when creating a DOS shell. This file has a .SWP extension. Do not delete this file or you will not be able to return to PLDshell Plus.

Modifying options for PLDshell Plus.

Quit — Exit to DOS.

Edit Menu

Figure 3-6 shows the Edit Menu. Use this menu to create or edit your PLDasm source files.



Figure 3-6 PLDshell Plus Edit Menu

Editor — Displays the currently defined text editor. A different editor can be selected using the **Change Editor** button. You can also temporarily set the editor by typing the editor program name in this field.

Filename — Displays the filename to be edited. The initial filename is *.PDS. The default extension of .PDS can be changed to any convenient three-character string. Pressing <Space> when *.PDS is displayed will display a list of all files with the .PDS extension. If no files exist in the current directory, an error message will be displayed. To enter a file with no extension, type the filename followed by a period (but no extension). To invoke a text editor with no filename, press <F6> to clear the field, then accept the menu. See the Utilities Menu to change the current directory or the default filename extension.

Change Editor — Allows you to select the editor of your choice. This button displays the Utilities—Options submenu with the cursor positioned on the Text Editor field. Type the command name of the desired editor in this field and press <F10>.

Compile/Sim Menu

Figure 3-7 shows the Compile/Sim Menu. Use this menu to compile and/or simulate a logic design.



Figure 3-7 PLDshell Plus Compile/Sim Menu

Input Filename — Displays the filename of the source file to be compiled and/or simulated. If the design has an extension other than .PDS, it can be compiled by using the Source Extension option in the Utilities—Options submenu to reset the default file extension. If no files are in the current directory, an error message will be displayed. See the Utilities Menu to change the current directory or the default filename extension.

Processing — Double-click the mouse or press the <Space> key to see a list of the five processing options:

- Compile Then Simulate
- Compile Only
- Simulate Only
- Estimate Only
- Estimate Then Simulate

Compile Then Simulate compiles the design and performs the specified simulation. The output will be a JEDEC file (if compilation was successful) and the simulation history.

Compile Only compiles the design with the specified compilation options. The result will be a JEDEC file, if the compilation was successful.

Simulate Only performs simulation as specified in the design file, but does not produce a JEDEC file. Note that if Compile Only is run on the same source file after a Simulate Only session, the compiler will use the existing simulation history file to generate test vectors for the JEDEC file.

Estimate Only produces an estimation report of possible target devices. No JEDEC file is produced.

Estimate Then Simulate produces an estimation report of possible target devices. Simulation is then performed as specified in the design file, resulting in a simulation history output. No JEDEC file is produced.

Select the desired processing mode by clicking on that mode. Alternatively, use the ↑ and ↓ keys to highlight the desired mode then press <Enter> to select.

NOTE:

Refer to Chapter 5 for a task-oriented discussion of compilation, estimation, and simulation.

NOTE:

The processing setting can be saved to your configuration file by selecting the **Compile Options** button and selecting the **Save Compile Options** button in the Compile Options submenu.

Compile Options — Selects the Compile Options submenu (Figure 3-8). Note that some option combinations are not legal. If you select an illegal combination of compile options, an error screen displays the legal combinations.

Simulation Options — Selects the Simulation Options submenu (Figure 3-9).

Compile Options Submenu

The Compile Options submenu, shown in Figure 3-8, allows you to set the compiler options:



Figure 3-8 PLDshell Plus Compile Options Submenu

Expand Equations — Default/Yes/No. The initial state is Default. Yes/No values will override any .PDS file settings of the *EXPAND* option. This option control if PLDasm will expand equations to the required SOP (Sum-of-Products) form to allow minimization, DeMorgan's inversion, and fitting. The option of not expanding is provided for designs with equations already in SOP form which the designer does not want altered in any way.

Minimize (Espresso) — Default/Yes/No. The initial state is Default. Yes/No values will override any .PDS file settings of the *MINIMIZE* option. PLDshell Plus uses the Espresso minimization algorithm, which is highly successful in reducing equations to the least number of product terms. The option for NO minimization is provided for the designer who does not want the equations altered in any way.

Automatic Inversion — Default/Yes/No. The initial state is Default. Yes/No values will override any .PDS file settings of the *INVERSION* option. When the default or Yes is used, equations will be inverted using DeMorgan's inversion rules if the inverted form will use a smaller number of p-terms. When the NO option is selected, SOP equations are left uninverted. However, the minimizer will compute the inverted

and non-inverted form and, if the inverted form is smaller, will prompt the designer to approve the use of the inverted form on an equation-by-equation basis.

D/T Selection (FLEXlogic) — Default/Yes/No. The initial state is Default. Yes/No values will override any .PDS file settings of the *DT_SYNTHESIS* option. The Default or Yes value is used with FLEXlogic devices to generate “T”-type flip-flop inputs whenever a “D”-type flip-flop is specified, and vice-versa. The fitter will then select whichever method uses fewer product terms. This increases the ability to fit the design. It is possible that the compile time and memory usage may increase due to the additional processing while generating alternate equations.

Error File — Create an error file, Yes/No. The initial state is Yes.

Report File — Create a report file, Yes/No. The initial state is Yes.

Auto Display Report — Automatically displays the compiler or estimator report file Yes/No. The initial state is Yes.

Pin Assignments — Pressing the <Space> key displays a list of the six fitter options:

Use Defaults, as specified in Design File—The fitter will use the defaults specified in the design file.

Use Design and Previous Pin Assignments, Abort on no Fit —The fitter tries to fit the design using the pin assignments in the source file and any previous compile efforts¹. The fitter will remove previous pin assignments if necessary. It will abort if any user assigned pin does not fit. *This is the default fitter option if none is specified in the source file.*

Use Design and Previous Pin Assignments, but Reassign if Needed — The fitter tries to fit the design using the pin assignments specified in the source file and any previous compile efforts¹. If the design does not fit, the fitter will make the necessary changes to the pin and node assignments via the fitter’s auto-fit algorithms.

Use Design Pin Assignments, but not Previous, Abort on no Fit — The fitter tries to fit the design using the pin assignments specified in the source file. The fitter aborts if any pin does not fit.

Use Design Pin Assignments, but not Previous, and Reassign if Needed —The fitter tries to fit the design using the pin assignments specified in the source file. If the design does not fit, the fitter will make the necessary changes to the pin and node assignments via the fitter’s auto-fit algorithms.

Ignore all Pin Assignments — The fitter ignores all specified pin assignments and uses its own auto-fit algorithms to attempt a design fit.

1. Saved in a .PIN file. A .PIN file is a PLDasm internal data file that stores the pins used from a previous compilation of a PLDasm file. This file is created when compiling a design, and is used by PLDasm to reduce pinout changes to incremental revisions of your design.

Fitter (FLEXlogic) — Use File Default/Normal Fit/Complete Fit. The initial state is Use File Default. Normal Fit and Complete Fit override any .PDS file *FITTER_ALGORITHM* option. See Chapter 5 for more details on how to use this option.

Save Compile Options — Saves the selected compiler options which will become the default start-up values until changed.

Chapter 5 discusses these options in greater detail and explains how and when you may use them. Not all options will work with every device.

Simulation Options Submenu

The Simulation Options submenu, shown in Figure 3-9, allows you to set the simulation options:



Figure 3-9 PLDshell Plus Simulation Options Submenu

Show Asynch. Events — Yes/No. Initial default is No. When set to Yes, vectors are generated for asynchronous events that occur during stabilization of each simulation cycle. This feature allows designers to more easily identify race conditions/glitches in combinatorial or fundamental mode sequential designs.

Max. Asynch. Events — Sets the maximum number of asynchronous events that can occur during stabilization of each simulation cycle before aborting simulation. This number can be any decimal integer in the range of 1 to 32,767. The initial default is 32.

Save Simulate Options — Saves the selected simulation options, which then become the new default start-up values until changed.

Chapter 5 describes these options in greater detail and explains how and when you may use them.

View Menu

Figure 3-10 shows the View Menu. With this menu you can view source files, error/log files, report files, vector/waveform files, or any other file. This menu provides facilities for viewing, not editing.

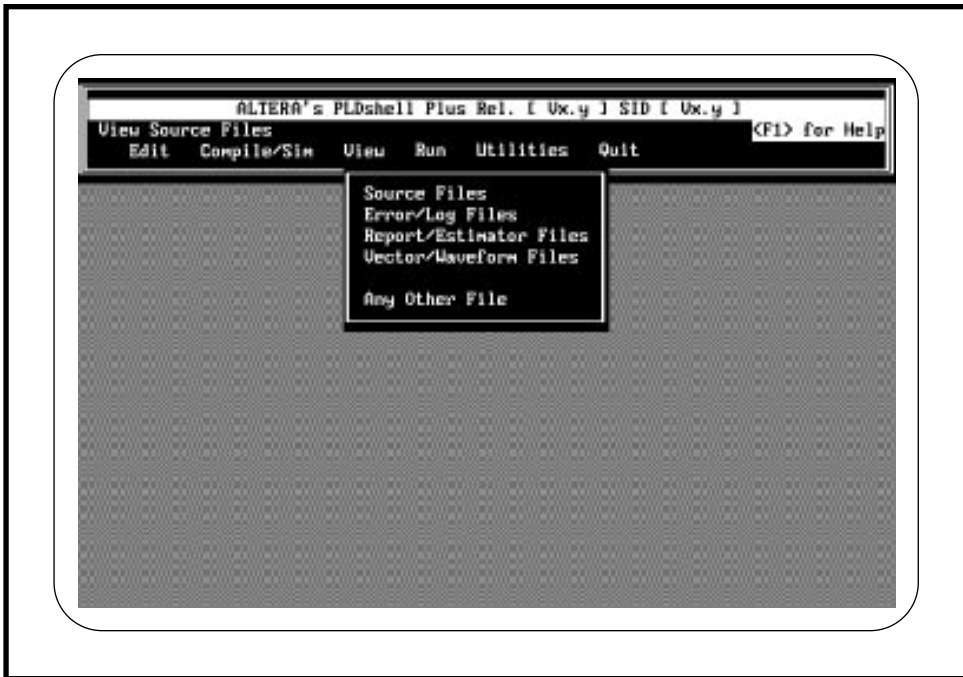


Figure 3-10 PLDshell Plus View Menu

Select an option with the mouse or use the ↑ and ↓ cursor keys to highlight a View option, then press <Enter> to select.

Source Files — Displays a list of source files to view. Use the ↑ and ↓ cursor keys to highlight a file, then press <Enter> to select. The default file extension is .PDS.

Error/Log Files — Displays a list of error/log files to view. Use the ↑ and ↓ cursor keys to highlight a file, then press <Enter> to select. The file extensions are .ERR and .LOG. When viewing an error file, you can move to a specific error/warning message and press <F10> to display on-line information to help you correct the error/warning.

Report/Estimator Files — Displays a list of report files generated by the compiler/simulator/estimator. Use the ↑ and ↓ cursor keys to select a file, then press <Enter>. The file extension is .RPT.

Vector/Waveform Files — Displays the View Simulation Vectors submenu (see Figure 3-11 and the discussion in the next subsection).

Any Other File — Allows you to view any other file in the current directory. The initial search pattern is for all files (*.*) .

Note that source, error, report, state table vectors, or other files are displayed in text format and are supported on all systems. The waveform viewer, however, can only run on systems with VGA, EGA, or Hercules graphics cards/monitors.

View Simulation Vectors Submenu

Figure 3-11 shows the View Simulation Vectors submenu. This menu selects the form in which vectors will be displayed and printed.



Figure 3-11 PLDshell Plus View Simulation Vectors

Input Filename — Displays the name of a simulation file to be viewed. The initial display is *.HST if more than one file is available. Pressing <Enter>, <F10>, or using the **Accept** button will display a list of all simulation files.

View Vectors as — Provides two viewing options: Waveform (Graphics) or as a State Table (1s and 0s). Use the <Space> key to toggle between the two options. Figure 3-12 shows an example of the graphics viewing option.

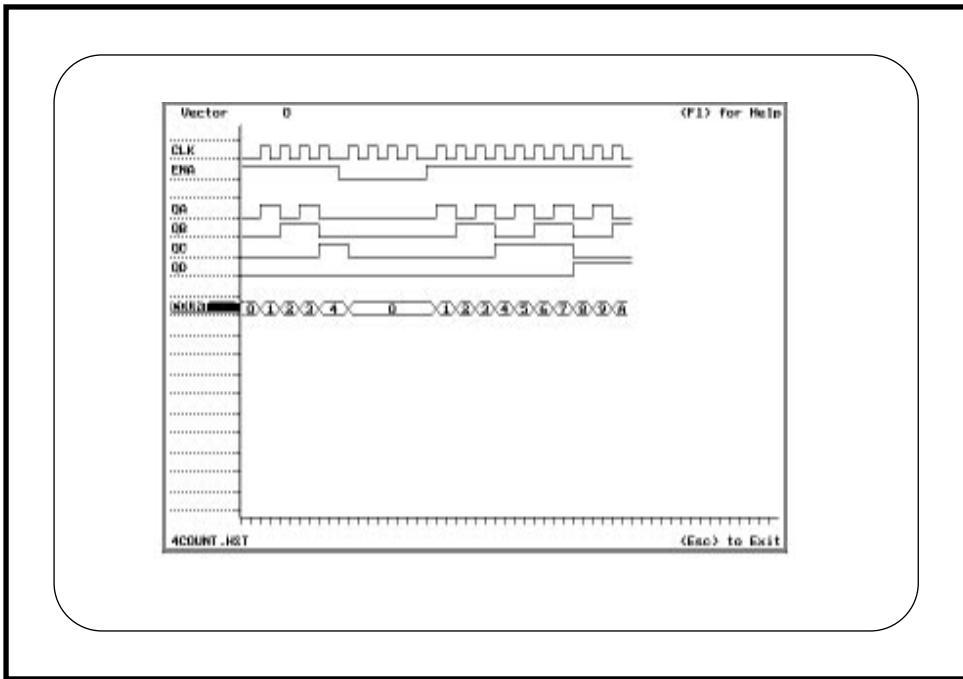


Figure 3-12 PLDshell Plus Sample Waveform Display

Print Vectors as — Provides two print options: Waveform (Extended ASCII) and Waveform (Plain ASCII). Waveforms can be printed using the line-drawing character set or with standard ASCII characters.

Print Page Length — Allows you to select the page length in terms of the number of lines. The default is 66 lines per page (6 lines per inch). Enter 0 for non-paged output (continuous feed). Printer output is to the print device defined in the Utilities —Options submenu. The default is .PRN.

Print Waveforms — Prints the waveforms using the format specified in the Print Vectors As field and the Print Page Length field.

Save View Options — Saves the currently defined View Simulation Vectors submenu options as the start-up options.

Graphics Waveform Viewer

PLDshell Plus provides PLDV, a powerful graphics waveform viewer, which supports:

- Zooming of the viewing area
- Moving, copying, or deleting a signal
- Cursor movement through either mouse or keyboard commands
- Ability to set a reference cursor to check relative timing
- Ability to save an edited waveform as a .HST file
- Ability to toggle between a waveform display and the corresponding .PDS file.

View Waveform Commands

The View Waveform screen responds to commands from both a mouse, if a driver is installed, and a keyboard. With a two-button mouse (left and right buttons only), some commands must be entered with the keyboard.

Mouse Functions

| | |
|------------------|---|
| Move Mouse Up | Moves cursor up one position If at top, moves signals down |
| Move Mouse Down | Move cursor up down position If at bottom, moves signals up |
| Move Mouse Left | Moves cursor left one position If at side, moves signals right |
| Move Mouse Right | Moves cursor right one position If at side, moves signals left |
| Left Button | If highlighted and held, drags signal |
| Left Button | If not highlighted, zooms in |
| Middle Button | If highlighted, copies signal |
| Middle Button | If not highlighted, toggles reference marker |
| Right Button | If highlighted, deletes signal |
| Right Button | If not highlighted, zooms out |

Key Pad Functions

| | |
|-------------------|---|
| Up Arrow (↑) | Moves cursor up one position If at top, moves signals down |
| Down Arrow (↓) | Moves cursor down one position If at bottom, moves signals up |
| Left Arrow (←) | Moves cursor left one position If at side, moves signals right |
| Right Arrow (→) | Moves cursor right one position If at side, moves signals left |
| Ctrl-Left Arrow | Moves cursor left half screen If at side, moves signals right |
| Ctrl-Right Arrow | Moves cursor right half screen If at side, moves signals left |
| Shift-Left Arrow | Moves screen left one position |
| Shift-Right Arrow | Moves screen right one position |
| Home | Moves cursor to left side of screen |
| End | Moves cursor to right side of screen |
| Ctrl-Home | Moves cursor to beginning of file |
| Ctrl-End | Moves cursor to end of file |
| PgUp | Drag signal up one position (if highlighted) Moves cursor up ten positions (if not highlighted) If at top, moves signals down |
| PgDn | Drags signal down one position (if highlighted) Moves cursor down ten positions (if not highlighted) If at bottom, moves signals up |
| + | Zoom in |
| - | Zoom out |
| * | Toggle reference marker |
| r | Reset program to starting conditions |
| ? | Display Help |

| | |
|---------------|--|
| Ins | Copy signal if highlighted |
| Del | Delete signal if highlighted |
| Esc or Ctrl-C | Exit program |
| Tab | Toggles between waveform display and source file |
| F1 | Displays the On-Line Help file. |
| Shift-F9 | Save current file. Up to the first five characters of the filename are used along with the suffix “-xx” plus the extension .HST. The number “xx” is in the range of 00 through 99. The name of the file being saved is displayed on the screen for 3 seconds. |
| F10 | Undo up to last five edits. If no edits have been made, the system beeps. |
| Shift-F10 | Invokes the File Browser. The file browse feature allows you to display up to eight consecutive text files using the same base filename as the current waveform file but with different extensions. Pressing <Shift-F10> the first time invokes the first file. Pressing <F10> after this sequences to the next file. Pressing <Shift-F10> after the first occurrence backs up one file. Use of the <Tab> key allows you to toggle between the current waveform and the current text file. |

Viewer Notes

- The current cursor position is displayed at the top left of the screen. The cursor position corresponds to the vector number in the .HST file.
- The reference marker is toggled on/off by the middle mouse button or the asterisk (*).
- When a reference marker is enabled, the delta position (i.e., the cursor position minus the reference marker position) is displayed next to the cursor position. Both of these fields change to red when zoomed out past one bit of information per pixel.
- The help screen can be exited at any time by pressing the <Esc> key instead of paging through the whole file.
- <PgUp> and <PgDn> keys allow you to move through the help file.
- To toggle between the waveform display and an ASCII text file (such as the source file for the design), use the **File Browse** command (<Shift F10>) to open the source file. Then use the <Tab> key to toggle back and forth.

Run Menu

Figure 3-13 shows the Run Menu. This menu allows you to run up to 24 preset application programs or batch files by pressing the appropriate letter key of <A> through <X>. There are two additional letters: <Y> and <Z>. <Y> allows you to run an application that is not one of the other 24 available keys. Pressing <Z> allows you to modify the Run Menu entries for letters <A> through <X>. Figure 3-14 shows the Modify Run submenu.

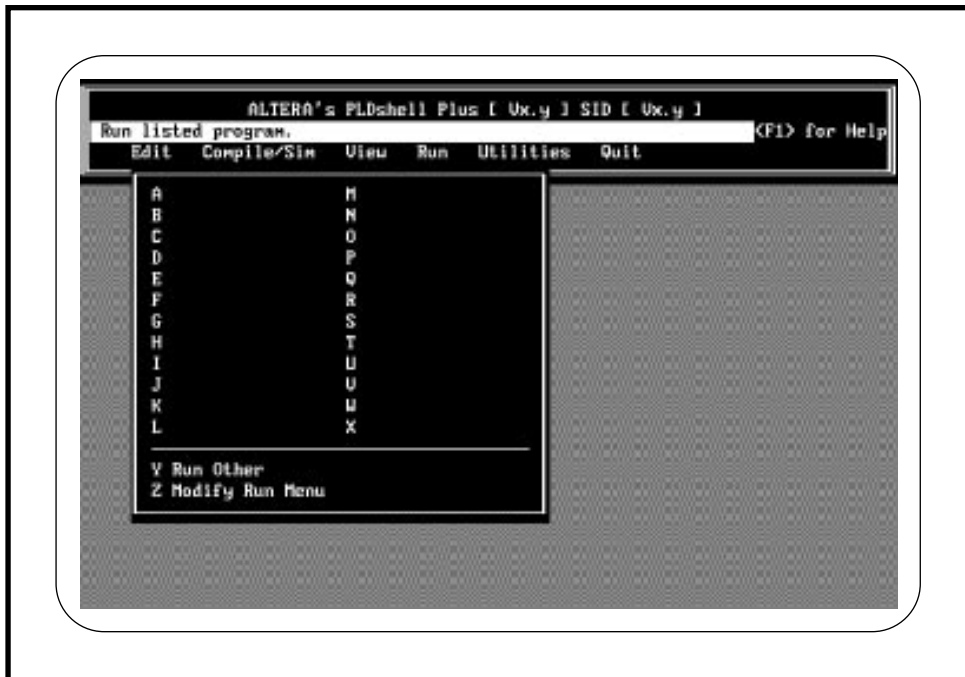


Figure 3-13 PLDshell Plus Run Menu

Modify Run Submenu

This submenu consists of three fields for each of the 24 application programs:

PROGx Label — Consists of a string of up to 16 characters that describe the application to be run.

Command — This is the command line to invoke the application program. The command line supports the following features to customize command lines:

- \$F uses the basename from previous processing
- \$D specifies the current working directory
- ? at the beginning of the command line results in user interaction before execution. This allows you to edit the command line if desired.

See “Run Menu Notes” for examples of how to use these features.

Run Dir — This is the DOS path for the application program.

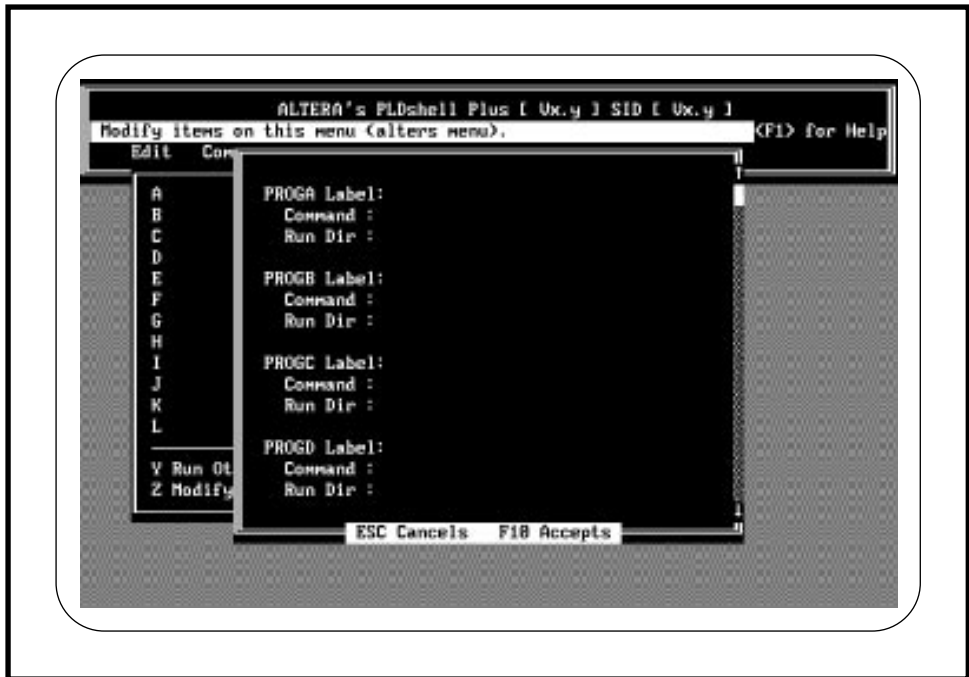


Figure 3-14 PLDshell Plus Modify Run Submenu

Run Menu Notes

1. To configure a Run Menu item, select the Run Menu, then select the Modify Run Menu option (<Z>). Move to a blank location and fill in the label (your choice), command line, and working directory fields. Press <F10> to accept the entry. You are prompted to save the changes. Answer “Y” to save the changes. Answer “N” if the changes are temporary.
2. The following sample Run Menu entry invokes a program that annotates JEDEC files:

```
PROGB Label: Annotate JEDEC  
Command: ?AJED $.JED device  
Run Dir:
```

This entry invokes a program named AJED, which requires the following command line:

```
AJED <filename>.JED <device name>
```

The label “Annotate JEDEC” will be displayed in the “B” position under the Run Menu since it is entered in the PROGB field. The “\$F” allows the current working base filename to be passed to the AJED program. A .JED extension is added by Run. The word “device” is a place-holder for the device part name, which must be filled in to run AJED properly. The “?” causes **Run** to pause before executing the command line; this pause provides the opportunity to fill in the device part name.

Since no working directory is specified, the current working directory is assumed. A “\$D” can be used in the command line if the working directory needs to be passed in the command line.

3. You can use the Run Menu to run the JED2JTAG and PENGN programs for configuring/programming FLEXlogic devices, and the APT program for programming Classic devices with Intel GUPI programming hardware. (For more information on JED2JTAG and PENGN, see Appendix F, FLEXlogic Prototyping Cable. For more information on APT, refer to your User’s Guide for an earlier version of PLDshell Plus/PLDasm.)
4. You cannot invoke Terminate-and-Stay-Resident programs (TSRs) from the Run Menu. You will encounter “Spawn Error” messages. In some cases, the system will hang, requiring a reboot. All TSRs should be loaded *before* invoking PLDshell Plus.

Utilities Menu

Figure 3-15 shows the Utilities Menu.

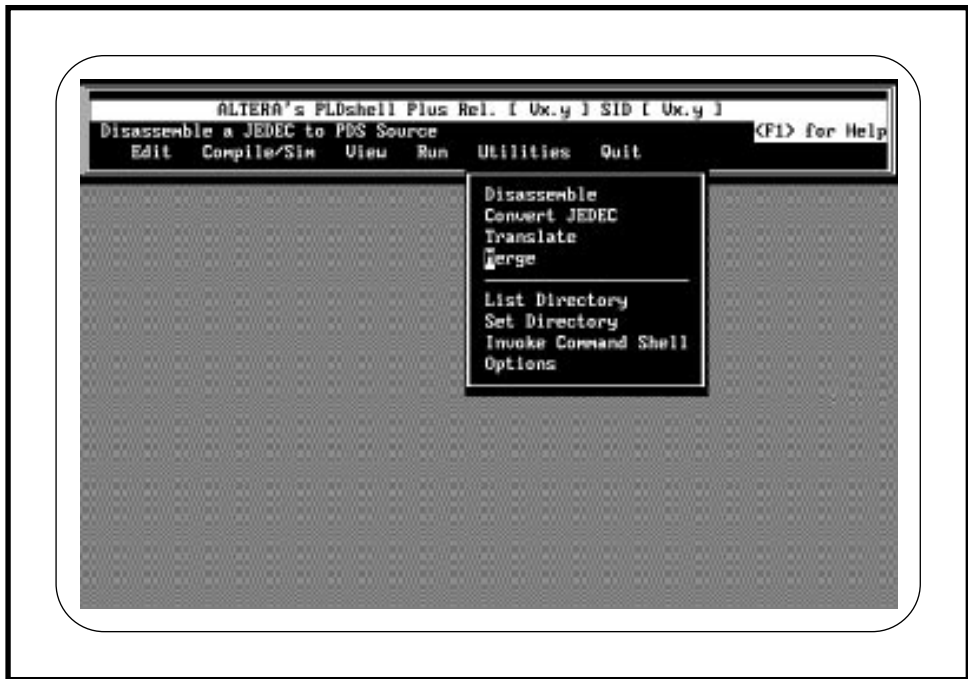


Figure 3-15 PLDshell Plus Utilities Menu

Disassemble — Disassembles existing JEDEC files (see Figure 3-16).

Convert JEDEC — Converts JEDEC files for common PALs/GALs into JEDEC files for Altera devices (see Figure 3-17).

Translate — Translates .ADF/.SMF source files to .PDS source files (see Figure 3-18).

Merge — Merges two to sixteen .PDS source files into a single .PDS file that can be compiled or estimated (see Figure 3-19).

List Directory — Lists the contents of the current directory.

Set Directory — Sets the working directory for PLDshell Plus.

Invoke Command Shell — Creates a DOS shell and displays the DOS prompt. To return to PLDshell Plus, type:

```
exit<Enter>
```

Options — Allows you to change default values for the following (see Figure 3-23):

- Command File
- Text Editor
- Print Device
- Menu Hotkeys on or off
- Source Extension

Disassemble Submenu



Figure 3-16 PLDshell Plus Disassemble Submenu

Input Filename — Displays a list of JEDEC files to disassemble. Click with the mouse or use the ↑ and ↓ cursor keys to select a file name, then press <Enter>. The default file extension is .JED.

Source Device — Sets the source device to be used. Double clicking with the mouse or pressing the <Space> key displays the list. Click with the mouse or use the ↑ and ↓ cursor keys to select a device type, then press <Enter>. The target Altera device will be displayed to the right of this field.

Package Type — Sets the package type. Double clicking with the mouse or pressing the <Space> key displays a list of package types for the device selected under Source Device. Click with the mouse or use the ↑ and ↓ cursor keys to select the package type, then press <Enter>.

Output Filename — Displays the output file name that will be created during disassembly. The default is the target device name plus .PDS.

Convert JEDEC Submenu

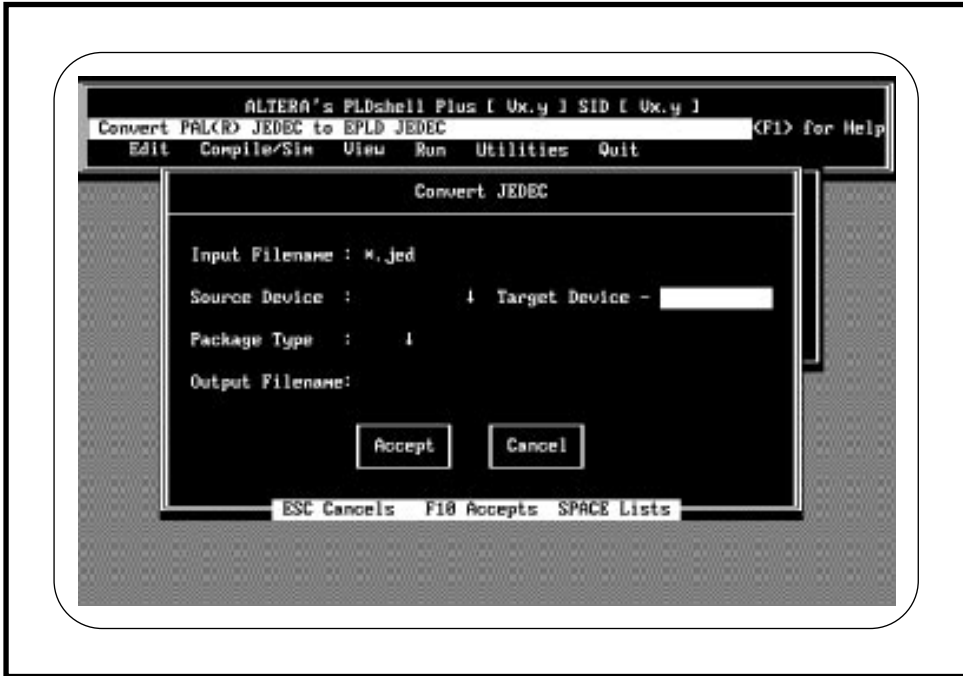


Figure 3-17 PLDshell Plus Convert JEDEC Submenu

Input Filename — Displays a list of JEDEC files to convert. Use the ↑ and ↓ cursor keys to select a file name, then press <Enter>. The default file extension is .JED.

Source Device — Sets the source device to be used. Pressing the <Space> key displays the list. Use the ↑ and ↓ cursor keys to select a device type, then press <Enter>. The target Altera device will be displayed to the right of this field.

Package Type — Sets the package type. Pressing the <Space> key displays a list of package types for the device selected under Source Device. Use the ↑ and ↓ cursor keys to select the package type, then press <Enter>.

Output Filename — Displays the output file name that will be created during conversion. The default is the target device name plus .PDS.

Translate Submenu



Figure 3-18 PLDshell Plus Translate Submenu

Input Filename — Displays a list of .ADF files to be translated. Use the ↑ and ↓ cursor keys to select a file, then press <Enter>. You can change the file type to *.SMF for A+PLUS State Machine Files.

Output Filename — Displays the file name selected under Input Filename, but with a .PDS extension. If there is a file name conflict, you can press <F6> to clear this field and type in a different file name. Use an extension of .PDS.

Merge Submenu

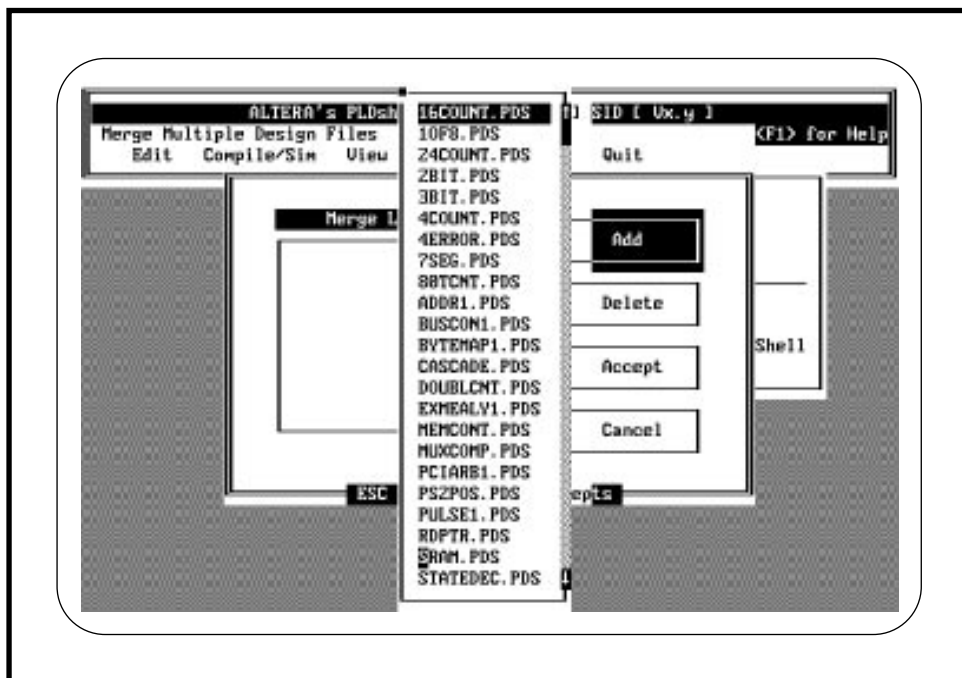


Figure 3-19 PLDshell Plus Merge Submenu

The Mergescreen has one field and four buttons:

Merge List — Displays a list of up to sixteen files to be merged. When first invoked, the list is empty. The minimum number of files is two.

Add — Adds a filename to the Merge List. Click on a filename in the file list to add a file to the Merge List. Only one file is added with each operation.

Delete — Deletes a filename from the Merge List. Displays a list of filenames that matches the Merge List. Clicking on a filename in the file list deletes a file from the Merge List. Deletes one file at a time. If only one file is in the Merge List, that file is deleted and the file delete window closes.

Accept — Accepts the files in the Merge List and displays the Use Original Pin Assignments screen (see Figure 3-20).

Cancel — Cancels the Merge Utility operation and returns to the Utilities Menu.

Use Original Pin Assignments Screen



Figure 3-20 Use Original Pin Assignments Screen

This screen displays the files in the Merge List (up to sixteen files). Each file is preceded by a pair of brackets.

Initially, the space between the brackets is blank. If you choose to use the original pin assignments for one or more files, click on the brackets and an “X” will appear. Click again to remove the “X.” The “X” indicates that the original pin assignments for a file will be used during the Merge operation.

Accept — Accepts the contents of this screen and displays the Resolve I/O Signal Names/Types screen (see Figure 3-21).

Cancel — Cancels the current operation and returns to the initial Merge screen with the filenames in the Merge List.

Resolve I/O Signal Names/Types Screen

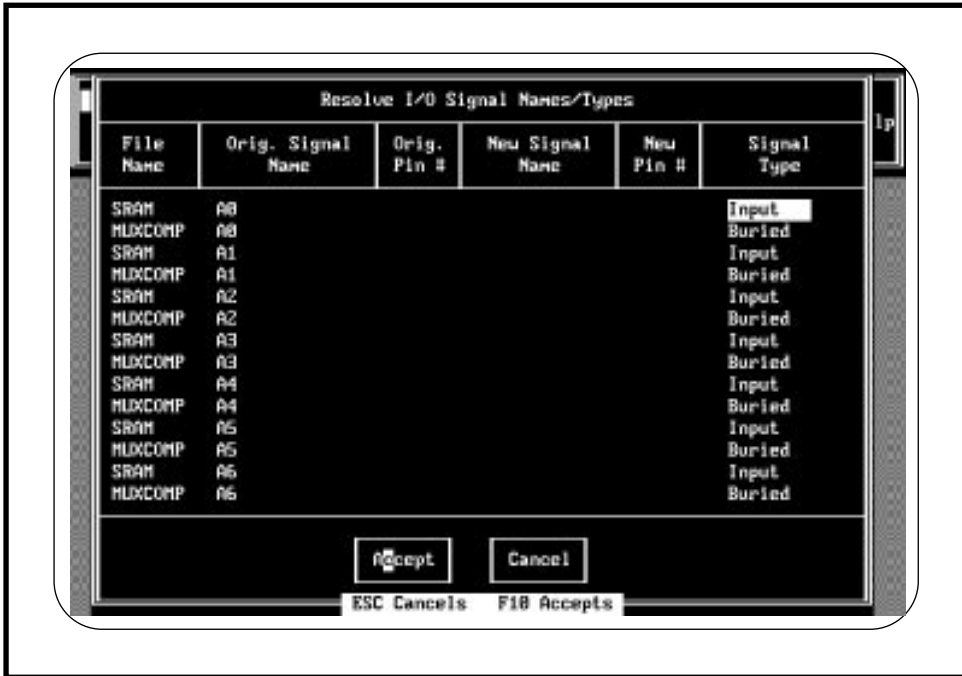


Figure 3-21 Resolve I/O Signal Names/Types Screen

This screen contains six columns. The three columns on the left cannot be edited; the three right-hand columns can be edited.

Source File — Filename of the source file for a particular signal.

Orig. Signal Name — Name of the signal in the source file.

Orig. Pin # — Pin number in the source file. If you chose not to use the original pin assignments, this field will be blank.

New Signal Name — Signal name in the merged design. This is determined by the user if a new name is desired.

New Pin # — The pin number in the merged design. The original pin numbers of a file, if selected, will be displayed. If no number is displayed, you can enter a pin number if desired. Alternatively, this field can be left blank and pin numbers can be assigned by the compiler.

Signal Type — Input, Output, or Buried. Inputs cannot be changed. You can toggle outputs between Output or Buried signal type via the <Space> key or mouse click.

You can use the arrow keys or <PgDn> and <PgUp> to scroll through the list.

Accept — Accepts the contents of the current screen and displays the Save Merged Design screen (see Figure 3-22).

Cancel — Cancels the current operation and returns to the initial Merge screen, with the filenames in the Merge List.

Save Merged Design Screen

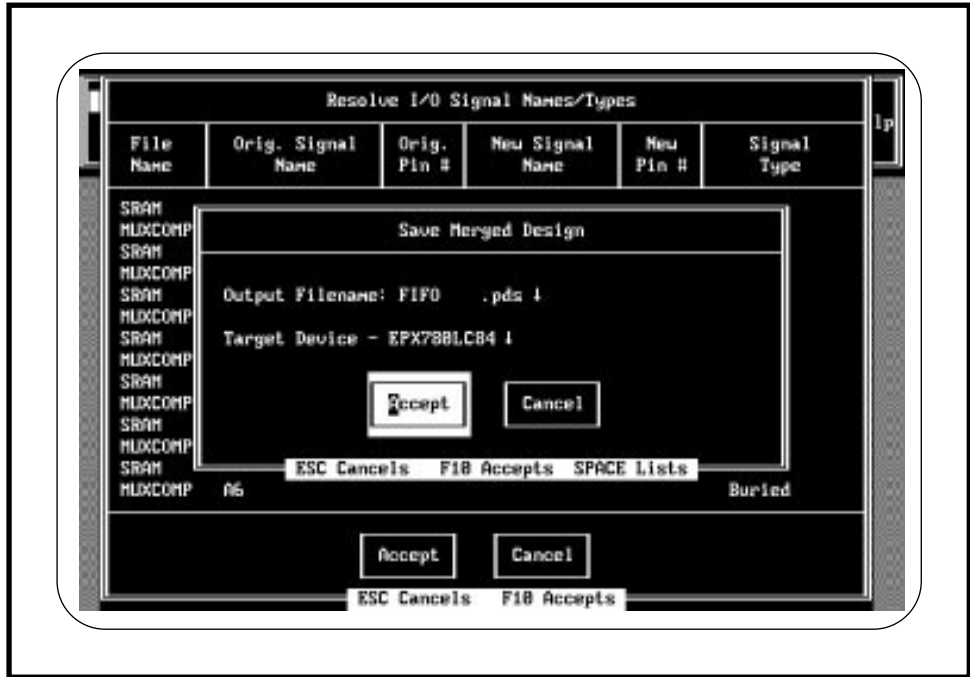


Figure 3-22 Save Merged Design Screen

This screen has two fields and two buttons.

Output Filename — This is the name of the merged .PDS file. The extension .PDS cannot be changed. Enter the name of the merged file in this field. You can press the <Space> key or double click in this field to display a list of .PDS files in the current directory.

Target Device — This is the target Altera device for the merge operation. Press the <Space> key or double click in this field to display a list of Altera devices. Select the target device using the arrow keys and <Enter> or click with the mouse.

Accept — Accepts the contents of the two fields and creates the merged .PDS source file. When the file has been successfully created, you will be prompted with an “OK” status screen.

Cancel — Cancels the current operation and returns to the Resolve I/O Signal Names/Types screen.

Options Submenu



Figure 3-23 PLDshell Plus Options Submenu

Command Shell— This is the name of the command file used by the operating system. For DOS, the default is COMMAND.COM.

Text Editor — Allows you to select the text editor used by the Edit Menu. The default is the editor you selected during installation (EDIT if you used the installation default). Can be set to your preferred editor. Arguments may be included in the invocation line. See “Run Menu Notes” for available arguments and how to use them.

Print Device — Allows you to set the printer device for use from the View Menu. The default for DOS is PRN. A filename can be entered to print to a disk file.

Menu Hotkeys — Sets the menu hotkeys On or Off. Using the <Space> key toggles between On and Off. The default is On. With hotkeys On, menu selections can be made using the first letter of the menu item. When set to Off, you must use the cursor keys and <Enter> to make a selection.

Source Extension — Sets the extension for the PLDasm source file. The default is .PDS. This option allows you to use a unique user or project extension for your designs.

Chapter 4 — PLDasm Files and Language

This chapter summarizes the structure of PLDasm files and describes the syntax for implementing PLD designs. It covers the following information:

- PLDasm Files
- Basic Circuit Design Using Boolean Equations
- Using Additional Features
- Truth Table Design
- State Machine Design
- Simulation

PLDasm Files

PLDasm files are standard ASCII files containing printable ASCII characters only. The minimum PLDasm file contains:

- Declaration section,
- One or more of the following Design sections
 - State Machine section (more than one acceptable)
 - Equations section (more than one acceptable)
 - Truth Table section
- Optional Simulation section

Figure 4-1 shows a summary of these PLDasm file sections.

```

; DECLARATION SECTION (REQUIRED) — Must be first in section.
; Header Information (Optional)
; Option Controls (Optional Section)
    OPTIONS
        TURBO = ON
        MINIMIZATION = OFF
; Chip Declaration (REQUIRED)
    CHIP        DESIGN_A    EP22V10
; Pin Declarations (REQUIRED)
    PIN    1    CLK
    PIN    2    IN1
    PIN    3    OUTA
; String Substitutions (Optional)
    STRING    SELA        '(INA * INB * /INC)'

```

3 DESIGN SECTIONS — At least one of these three sections is required:

```

; STATE MACHINE SECTION (Optional)
MACHINE MEALY_MACHINE
    S1 := BUS_REQ      ->    S2
    S2 := BUS_CONT     ->    S3
    + BUS_WAIT        ->    S4

```

```

; EQUATIONS SECTION (Optional)
EQUATIONS
    OUTA = IN1 * IN2 * /IN3
    OUTB := IN4 * IN5
    + IN6 * /IN5

```

```

; TRUTH TABLE SECTION (Optional)
T_TAB    ( in1  in2  >>  /out1  out2 )
          0   0   :    0     0
          0   1   :    1     0
          1   0   :    0     0
          1   1   :    0     1

```

```

; SIMULATION SECTION (Optional) — Must be last section
SIMULATE
    SETF  OE    /CLK  IN1   IN2
    PRLDF          /Q2  /Q1   /Q0
    SETF  IN1
    CLOCKF        CLKF

```

Figure 4-1 PLDasm File Section Summary

Comments

Comments can be included on any line. Comments begin with a semicolon (;). Everything between the semicolon and the end of a line is considered a comment.

The following are three examples of valid comments.

```
;PIN
  PIN 3   INB   ; THIS COMMENT FOLLOWS A PIN NAME

; THIS COMMENT IS ON A LINE BY ITSELF
```

Legal Signal Name Characters

Signal names can include 1 to 14 alphanumeric characters (A–Z, a–z, and 0–9) and underscore (_). PLDasm is *not* case-sensitive. *The first character must be an alphabetic character.* Any other printable ASCII character is illegal and will cause errors during compilation if they appear in signal names.

Declaration Section (Required)

The first section in all PLDasm files is the Declaration section (see Figure 4-2); it is required. The Declaration section provides a means to identify designs and to track revisions. It contains the following six fields which may appear in any order:

- Header Comment (optional)
- OPTIONS Declaration (optional)
- CHIP Declaration (required)
- PIN Declarations (required)
- STRING Substitutions (optional)
- FUSE Statements (optional) —use with extreme caution

Each field starts with its respective keyword and can contain a full line of text.

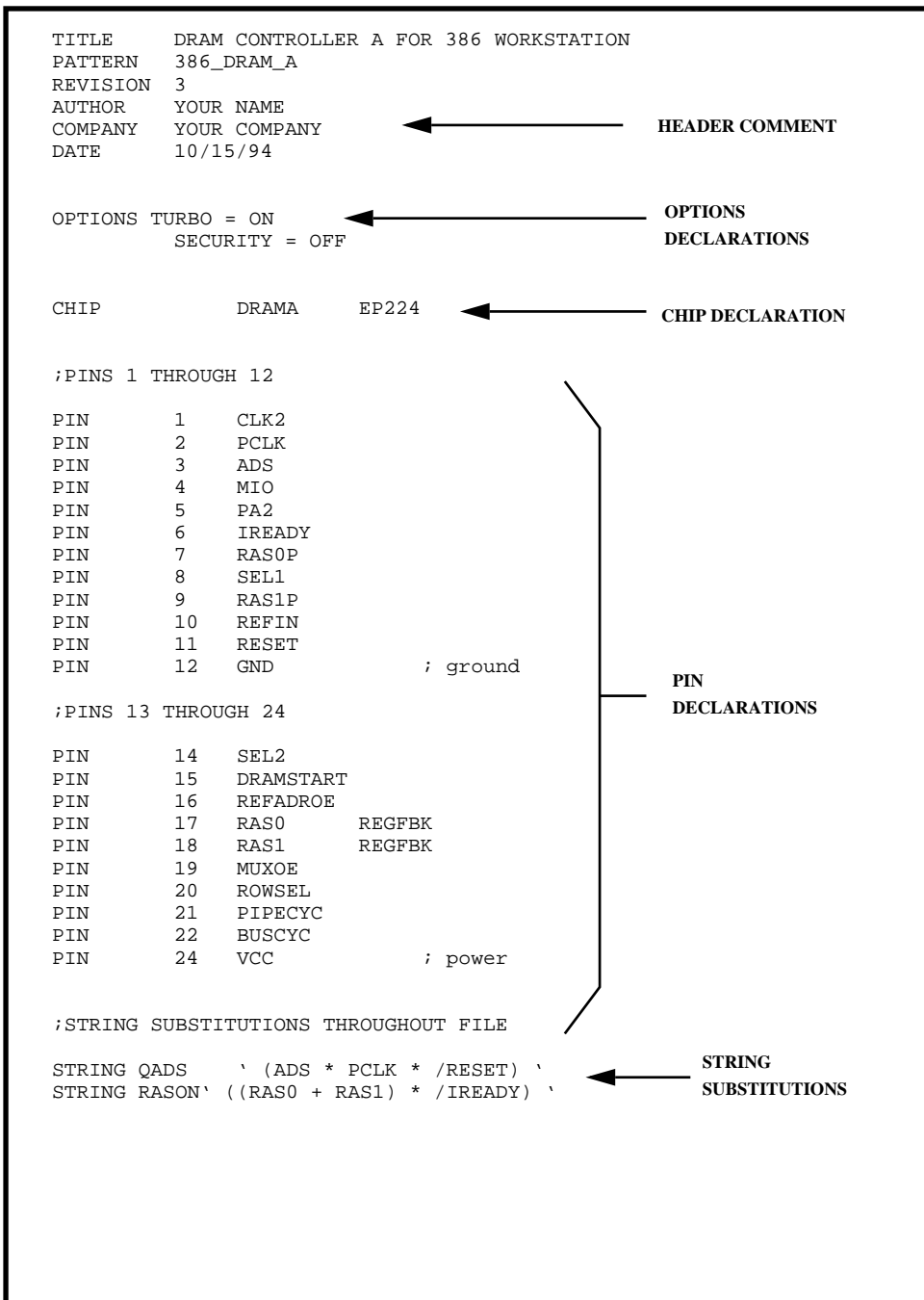


Figure 4-2 Example Declaration Section

Header Comment (optional)

The entire Header is actually a comment. The Header Comment is composed of the following fields:

- Title Field — Name of the design.
- Pattern Field — Pattern number.
- Revision Field — Version number.
- Author Field — Designer's name.
- Company Field — Name of your company.
- Date Field — The date of the design.

OPTIONS Field (optional)

The OPTIONS field allows you to specify several compiler and device controls. The syntax is shown here for all of the options, with a brief description. It must be *before* the CHIP keyword. Not all options are available for every device. The default value is the first value in each options list.

- TURBO = [ON|OFF] ; sets TURBO bit in device, if any
- SECURITY = [OFF|ON] ; sets SECURITY bit(s) in device, if any
- EXPAND = [ON|OFF] ; Reduce equations to 2-level SOP equations
- INVERSION = [ON|OFF] ; Allow left-hand side of equations to be inverted
- DT_SYNTHESIS = [ON|OFF] ; D or T Flip-flops; FLEXlogic devices only
- MINIMIZATION = [ON|OFF] ; Controls Espresso algorithm

Options that work only with the FLEXlogic device fitter are as follows:

- DRIVE_LEVEL = [5VOLT|3VOLT] ; default I/O pin voltage level
- FITTER_PINS = [KEEP|TRY|IGNORE] ; what to do with pins in .PDS file
- FITTER_PREV_PINS = [ON|OFF] ; use .PIN file first, if it exists
- FITTER_ALGORITHM = [NORMAL|COMPLETE] ; COMPLETE requests extended (i.e., exhaustive) fitting. See full description below.

The keywords (i.e., SECURITY) and either ON or OFF are required, as is the equal sign (=).

When using the PLDshell GUI interface, all of these options can be overridden by the Compile Options submenu, except for TURBO, SECURITY, and DRIVE_LEVEL. A detailed description of each option is given.

If the options are not specified in the file, PLDshell/PLDasm will use the default values and list those it has used in the design report file.

TURBO=[ON|OFF] (default: ON)

Many Altera devices supported by PLDshell Plus provide a Turbo Bit that allows you to optimize a design for speed or for power savings. When the Turbo Bit is on (TURBO=ON), the device is optimized for speed. When the Turbo Bit is off (TURBO=OFF) the device is optimized for power savings and will enter standby mode in low frequency applications if no transitions are detected for a period of time. Refer to the component data sheet for additional information on the Turbo Bit.

SECURITY = [OFF|ON] (default: OFF)

Most Altera PLDs provide a Security Bit (or Verify Protect Bit) that, when programmed, prevents the contents of the device from being read. This feature provides design security. If the state of the Security Bit is not specified, PLDasm automatically sets the Security Bit to OFF to allow design information to be read.

EXPAND = [ON|OFF] (default: ON)

This option controls the PLDasm equation expansion process, which will reduce all equations (removing parentheses, distributing strings, etc.) to 2-level SOP equations. Only set EXPAND=OFF if you have reduced the equations yourself, and there are no parenthesized sub-expressions. This affects *all* equations in the file.

INVERSION = [ON|OFF] (default: ON)

This option allows PLDasm to perform a DeMorgan's operation, which will invert the left-hand side of an equation, and affects *all* equations in the file. Note that this inversion may be desirable only for certain specific logic systems such as asynchronous designs. Only set INVERSION=OFF if you require the equations to not be changed. Disabling inversion will also limit the possibilities of the minimizer to choose a smaller version of some equations.

DT_SYNTHESIS = [ON|OFF] (default: ON)

This option is for FLEXlogic devices only. PLDasm will generate the corresponding D or T equation for a flip-flop, minimize them, and then pick the smaller one. On machines with limited memory, you can set DT_SYNTHESIS = OFF to use less memory and to speed up the compilation process.

MINIMIZATION = [ON|OFF] (default: ON)

This option controls whether the EspressoII algorithm is used, and affects *all* of the equations in the design. Set MINIMIZATION=OFF if you do not want minimization, but watch for fitter errors from equations that may be too large.

DRIVE_LEVEL = [5VOLT|3VOLT] (default: 5VOLT)

This option is for FLEXlogic devices only. This option controls the default I/O pin voltage level for those devices that do not have it specified in the PIN declaration with the 3VOLT or 5VOLT keywords. (Example: PIN 3 FOOB 5VOLT).

FITTER_PINS = [KEEP|TRY|IGNORE] (default: KEEP)

This option tells the fitter what to do with pins in .PDS file. KEEP means do *not* move the pins around. TRY means attempt to use the pin assignments given, but move them if necessary. IGNORE means that the fitter will not use the input file pin assignments, if any, and will generate its own instead. This option affects all the specified pins in the design.

FITTER_PREV_PINS = [ON|OFF] (default: ON)

This option is for FLEXlogic devices only. When set to ON, this option tells the fitter to use the assignments found in the .PDS file first, then the .PIN file, if it exists. The .PIN file is created by the fitter every time it is run, and is used to preserve the last fitting attempt. In this fashion, incremental changes to your design should not disturb already fitted portions. If there are problems with fitting, deleting the .PIN file to get a fresh assignment might be useful. Also, copying the pin assignments from a previous report file into the .PDS file can be useful.

FITTER_ALGORITHM = [NORMAL|COMPLETE] (default: NORMAL)

This option is for FLEXlogic devices only. The fitter will use NORMAL most of the time. Only use FITTER_ALGORITHM=COMPLETE when your design is not fitting. COMPLETE can take a long time! Before trying COMPLETE mode, make sure that your clock pins are assigned, and assign as many of the I/O pins as you can. Also, deleting old .PIN files if things have been changed substantially may help as well.

CHIP Field (required)

The CHIP field is the most important field. If it does not appear in the design, the entire file is treated as a comment. It contains the following sub-fields:

- Design Name
- Target Device Name

The CHIP field begins with the keyword CHIP and is followed by the design name and then by the device name. The CHIP sub-fields are defined as follows:

Design Name

The design name is required. It can be any name.

Device Name

The device name is required for device-specific designs and must be one of the devices supported by PLDshell. For device-independent design, the reserved keyword ALTERA_ARCH can be used. Chapter 5 describes the differences between device-specific and device-independent design.

PIN Names (optional)

Names the input and output signals. The names can include 1 to 14 alphanumeric characters; the first character must be an alphabetic character. The first example shows the easiest method of defining pins. The order of the pins is not important with this method.

```
PIN 1    ADDR0
PIN 2    ADDR1
PIN 4    ADDR3
PIN 5    ADDR4
PIN 3    ADDR2
```

When using this method of pin declarations, a keyword can be added after the signal name to define the architecture of I/O pins on devices that offer additional I/O options. This is described later in the “Using Additional Features” section.

PLDshell Plus is capable of automatically assigning inputs and outputs to pins in many cases. Automatic pin assignment can be implemented by omitting the pin *numbers* in the declaration section of the source file. Since the device and package are known to PLDasm, the PLDasm fitter uses its internal algorithms to fit designs to the target devices. The recommended approach is to specify all control signals for all equations in the design, including dedicated clock pins.

STRING Field (optional)

Allows global string substitutions in PLDasm files. Each string substitution begins with the keyword **STRING** followed by the name of the string and the substitution value. In the example in Figure 4-2, string substitutions are used to group some signals that are frequently used together. This can help simplify the Equations section. It is recommended that all text in substitution strings be enclosed in parentheses to improve readability in output files. This will ensure correct results when used with other Boolean operators.

FUSE Statement (optional)

This statement should be used with extreme caution. It is for expert users only. **FUSE** allows you to set specific JEDEC fuses by address. The **FUSE** statement is used as follows:

```
FUSE      0xf8    : 0
sets JEDEC bit at hex address f8 to zero.

FUSE      [0X100:0x200] : 1
sets JEDEC bits in the address range 100 to 200 hex to 1.
```

Basic Circuit Design Using Boolean Equations

This section describes basic combinatorial and registered circuit design for Altera PLDs supported by PLDshell Plus using Boolean equations. Advanced registered design topics, such as asynchronous clocking of registers, and use of T-type, JK-type, and SR-type flip-

flops, Automatic Pin Assignments, Alternate I/O Options, Truth Tables, and State Machines are covered in later sections.

A complete list of PLDasm operators (combinatorial and registered) is as follows, in order of precedence:

| Operator | Description |
|-----------------|--|
| / | Active-Low in pin declaration; Boolean NOT elsewhere in file |
| * | Boolean AND |
| :+: | Boolean XOR |
| + | Boolean OR |
| = | Combinatorial Output |
| *= | Latched Output |
| := | Registered Output |

Combinatorial Circuits

Combinatorial circuits are easily implemented in PLDasm files using Boolean equations. These equations must be placed in the Equations section of PLDasm files. An Equations section begins with the keyword EQUATIONS. The output from the equation is a pin name or node name elsewhere in the design. Figure 4-3, Combinatorial Circuits Using Boolean equations, shows some simple logic functions implemented using Boolean equations.

The first seven equations each implement basic logic gates. The eighth equation (SUM1) is a slightly more complex logic function with three AND terms (product terms, or p-terms) feeding one OR gate. The last equation (SUM2) shows how the output from one equation can be fed back and used to qualify other equations. In this example, the SUM1 output is fed back through the logic array to help qualify SUM2.

Note that parentheses can be used to specify precedence in Boolean equations. SUM1 shows an example of this. Equations can be expressed in Sum-of-Products form, but this is not a requirement.

```

EQUATIONS

AND1      = IN1 * IN2      ; LOGICAL AND

NAND1     = /(IN1 * IN2) ; LOGICAL NAND

OR1       = IN1 + IN2     ; LOGICAL OR

NOR1      = /(IN1 + IN2) ; LOGICAL NOR

A         = /IN          ; LOGICAL NOT

XOR1      = IN1 :+: IN2   ; EXCLUSIVE OR

XOR2      = IN1 * IN2 :+: INA * INB + INC

          ; EXCLUSIVE OR - PRECEDENCE OF ABOVE STATEMENT IS
          ; ((IN1 * IN2) :+: (INA * INB)) + INC

SUM1      = IN1 * IN2 * IN3
          + /IN1 * (/IN2 * IN4)
          + IN1 * IN5 * IN6; SUM OF PRODUCTS

SUM2      = IN1 * IN2 * /SUM1
          + IN1 * /IN2 * /SUM1
          + /IN1 * IN5 * /SUM1; SUM OF PRODUCTS

```

Figure 4-3 Combinatorial Circuits Using Boolean Equations

Active-High/Active-Low Outputs

Macrocells on the Altera devices supported by PLDshell contain an inversion control bit that allows the respective output to be configured as active high or active low. An output is configured as active low when *either* the pin name *or* equation feeding the pin includes the slash prefix (/). An output is configured as active high when the polarities of *both* the pin name *and* equation match, i.e., both do not have the slash prefix. In the examples in Figure 4-4, Active-High and Active-Low output equations, OUTA and OUTB are both active-low outputs. OUTC is active high. OUTD is also active high. When slashes are used in the pin name *and* equation feeding the pin, the result is active high.

```

PIN      10      OUTA
PIN      11      /OUTB
PIN      12      OUTC
PIN      13      /OUTD

EQUATIONS

/OUTA = A * B * C * /D      ; active low output

OUTB = A * B * C * /D      ; active low output

OUTC = A * B * C * /D      ; active high output

/OUTD = A * B * C * /D      ; active high output
                               ; polarities cancel each
                               ; other

```

Figure 4-4 Active-High and Active-Low Output Equations

NOTE:

Output levels are affected by conversion. See Chapter 5, Device-Dependent Design Notes, for further information on conversion.

An alternate method for specifying an output as active high or active low is to use the HIGH or LOW keyword in the pin declaration. This option is shown below:

```

PIN      11      OUTB LOW
PIN      10      OUTA HIGH

```

NOTE:

This option is not supported with the older order-specific method of pin declaration.

Output Enable

The .TRST extension is used to specify controlling logic for the OE. The available logic varies from device to device. An example of the usage is:

```

OUTB.TRST = E * G * /D      ; output enable for OUTB

```

For more information on the usage of output enable for a specific device, see the relevant data sheet in the current Altera Data Book.

Registered Circuits

Registered circuits are easily implemented using the same Boolean equation syntax as combinatorial circuits, but with a “:=” operator in place of the “=”. This syntax implements a standard D-type register circuit.

The following is a list of all valid signal extensions and their descriptions:

| Extension | Description |
|------------------|---|
| .ACLK | Asynchronous Clock |
| .CLKF | Clock Pin (Synch.) or Clock Equation (Asynch.) |
| .ALE | Asynchronous Latch Enable |
| .LE | Synchronous Latch Enable |
| .TRST | Output Enable Equation |
| .RSTF | Clear Equation |
| .SETF | Preset Equation |
| .D | Data Input to D-Type Register |
| .T | Data Input to Toggle Register |
| .J | J Data Input to JK Register |
| .K | K Data Input to JK Register |
| .R | R Data Input to SR Register |
| .S | S Data Input to SR Register |
| .FB | Feedback Path from I/O Pin (sometimes needed during simulation) |
| .ADDR | RAM Address Line |
| .DATA | RAM Data Line |
| .BE | Block Enable |
| .WE | Write Enable |
| .CMP | Compare Term |

When using the dedicated clock pin on a PLD with a single clock, it is not necessary to specify the clock input. PLDasm will use the dedicated clock by default. You *may*, however, choose to specify the clock as part of your design methodology. The clock for the circuit uses the output name with a .CLKF extension on the left-hand side of the “:=” symbol and the clock input name on the right-hand side. Figure 4-5, Registered Circuits Using Boolean Equations, shows several examples of registered circuits.

The first register, QOUTA, is clocked by the dedicated clock pin (since no clock is specified) and is always enabled (OE tied to VCC).

The second register, QOUTB, is also clocked by CLK. Its output buffer, however, is controlled by the OE input signal. The clock is specified here.

The third register illustrates the use of register feedback in designs. This circuit uses feedback from the first two registers as its input. It is clocked by CLK (not specified) and its output buffer is controlled by a p-term.

On devices that have more than one clock pin (e.g., the EP600 has two clock pins), you must specify the clocks to avoid conflicts. The fourth example shows this (OUT3 and OUT13).

Some devices, such as the EP22V10, EP22V10E, EPX8160, EPX780, and EPX740, have a programmable synchronous clock inversion option to allow registers to be clocked on either the rising or falling edge of the global clock signal. The fifth example shows outputs clocked by the non-inverted and inverted clock (OUT4 and OUT5).

```

; OUTPUT, CLOCK, AND OE PINS

PIN      CLK
PIN      CLK2
PIN      OE
PIN      QOUTA
PIN      /QOUTB
PIN      QOUTC
PIN      OUT4
PIN      OUT5
PIN      QOUT3
PIN      /QOUT3
PIN      QOUT13
PIN      /QOUT13

EQUATIONS

; D-register with default clock and OE always enabled

QOUTA:= IN1 * IN3
        + IN1 * /RESET
QOUTA.TRST = VCC
; D-register with clock specified and OE controlled
; by an input pin (OE)

QOUTB   := INA * /IN1
QOUTB.CLKF = CLK
QOUTB.TRST = OE

; D-register with default clock, output controlled by
; p-term, equation includes feedback from other
; registers

/QOUTC   := QOUTA * /QOUTB
QOUTC.TRST = IN1 * OE

; Two D-registers, each controlled by different clocks.
; Supported on devices with multiple clocks or that
; support asynchronous clocking.

/QOUT3:= INA * INB
QOUT3.CLKF = CLK

QOUT13:= INA * /INB * IND
QOUT13.CLKF = CLK2

; Two D-registers, one clocked by CLK
; and the other by CLK inverted.

OUT4 := IN1 * IN4 * INA
OURT4.CLKF = CLK
OUT5 := IN2 * INB
OUT5.CLKF = /CLK

```

Figure 4-5 Registered Circuits Using Boolean Equations

Using Additional Features

This section covers design of circuits that make use of the additional features provided by the Altera PLDs supported by this software. Topics include asynchronous clocking of registers, use of the Preset and/or Clear p-term, and use of T-type, JK-type, and SR-type registers. Also described is the syntax for using global resources and for taking advantage of some of the more complex I/O architectures in the previously mentioned devices. The features described here are not available on all devices.

Asynchronous Clocking

Asynchronous clocking is available on many Altera PLDs. Asynchronous clocking of registers is implemented by assigning the register clock to an equation or pin other than a dedicated clock input. This allows logic functions to be used as register clocks. For example, each register in the EP610 and EP600 contains an OE/Asynchronous Clock p-term, which allows each of the 16 registers to be clocked independently.

The first example in Figure 4-6, Extended Register Options, shows a simple equation being used to clock QOUTD. Note that the clock signal uses the output name with a .ACLK extension. A single p-term equation is then specified as the clock signal. The output buffer is tied to VCC (always enabled). Use of the register name with an .ACLK extension is the recommended method for specifying asynchronous clocks.

An alternate method for specifying asynchronous clocks is to use a .CLKF extension. Note, however, that the PLDasm compiler will assign a clock signal using this extension to a dedicated synchronous clock pin if the clock signal is (1) driven by a single active-high input signal, and (2) the input signal is unassigned or inadvertently assigned to a pin that can function as a dedicated clock. For sake of consistency, the .ACLK extension is recommended.

Preset P-Term

Asynchronous Presets are available on each macrocell in some devices to allow registers to be independently preset (to 1) when the specified equation is true (high). The Preset equation is implemented by using the output name with a .SETF extension on the left-hand side of the “=” sign and an equation on the right-hand side. The third example in Figure 4-6 shows an example of a single p-term preset equation.

Clear P-Term

An asynchronous Clear p-term is available on many devices to allow registers to be independently reset (to 0) when the specified equation is true (high). The second and fourth

examples in Figure 4-6 make use of this feature. Both examples use a single p-term equation to clear a register. The Clear function is implemented in an equation by using the output name with a .RSTF extension on the left-hand side of the “=” sign and an equation on the right-hand side.

```

;OUTPUT, CLOCK, AND OE PINS

CLK1 OE CLK2 QOUTD /QOUTT QOUTK QOUTR SET

EQUATIONS

; D-register with asynch. clock, OE always enabled

QOUTD      := INA * INB
            + INA * INC * /IND * /RESET
            + /QOUTD * INC * INE
QOUTD.ACLK = IN1 * IN2 * /IN4
QOUTD.TRST = VCC

; Asynchronous clock could also be implemented as follows:
;
; QOUTD.CLKF = IN1 * IN2 * /IN4

; T-register with synch. clock, asynch. reset, OE control

QOUTT.T     := INA * /INC
QOUTT.CLKF  = CLK1
QOUTT.RSTF  = IN1 * IN2 * RESET
QOUTT.TRST  = OE * /RESET

; JK-register with synch. clock, OE control, preset

QOUTJ.J     := QOUTD * /INA
QOUTJ.K     := INB * INC * /QOUTD
QOUTJ.CLKF  = CLK1
QOUTJ.TRST  = IN1 * OE
QOUTJ.SETF  = SET

; SR-register with synch. clock, asynch. reset, OE always
; enabled

QOUTR.R:= QOUTA * IN1 * IN2 * /QOUTB
QOUTR.S:= /QOUTA * INA * INB
            + /QOUTB * IN1 * IN2
            + RESET * /IND
QOUTR.CLKF  = CLK1
QOUTR.RSTF  = IN1 * IND
QOUTR.TRST  = VCC

```

Figure 4-6 Extended Register Options

Toggle Flip-Flops

A T-type, or toggle, flip-flop is available on many devices and is implemented as shown in the second example in Figure 4-6. QOUTT is the output pin name. A .T extension is added to the output name to designate the register input. QOUTT is clocked by a synchronous clock (dedicated clock pin) and includes an output enable p-term and a reset p-term.

JK Flip-Flops

QOUTJ is the output pin name for a JK-type flip-flop. The “J” input is identified by using the output name with a .J extension. In the same way, the “K” input uses the output pin name with a .K extension. Note that this is a clocked emulation of an RS flip-flop, not a true asynchronous circuit. (See the notes after “SR Flip-Flops.”) QOUTJ also includes an output enable p-term.

SR Flip-Flops

QOUTR is the output pin name for an SR-type flip-flop. The “S” input is identified by using the output pin name with a .S extension. In the same way, the “R” input uses the output pin name with a .R extension. Note that this is a clocked emulation of an RS flip-flop, not a true asynchronous circuit. QOUTR also includes a clear p-term and the output buffer is tied to VCC (always enabled).

NOTES:

Altera devices supported by PLDshell Plus emulate JK and SR flip-flops as synchronous registers. The registers do not change state until the clock signal changes.

When JK and SR flip-flops are selected, the product terms are shared among two OR gates (one OR gate for the J or R input, and one OR gate for the K or S input). The allocation of these product terms may be described as follows:

$$\begin{aligned} \text{J or R} &= n \\ \text{K or S} &= (\text{available terms} - n) \end{aligned}$$

Using Global Set/Reset Signals

Some devices contain global signals, such as a register set and a register reset. The EP22V10, EP22V10E, and EP310, for example, both contain a synchronous set p-term and an asynchronous reset p-term. The set and reset signals can be driven by the true or complement of any input or I/O pin, or by any logic equation that can be implemented in a

single AND expression. Figure 4-7, Global Set/Reset, shows an example of these global signals, along with the PLDasm syntax to make use of them.

Note that a virtual pin (pin 25 on the EP22V10 and EP22V10E and pin 21 on the EP310) is used to identify the global signal, GLOBAL in this case. This name is then in the equations section, along with the .SETF and .RSTF extension to specify the logic for the set and reset signals, respectively. In the example, the synchronous set is connected to the SET input (pin 2). All registers in the device will be set (preset) to a logic one when SET and ENABLE are high and the registers are clocked. The synchronous reset is connected to the complement of the RESET signal (pin 3). All registers in the device will be reset (cleared) to zero when RESET goes low and ENABLE goes high. Note that the programmable inverter in the EP22V10, EP22V10E, and EP310 is located at the output of the register. When using register set and reset to set the register output, the inverter in these devices (when used) will invert the output of the register from the internal high or low state.

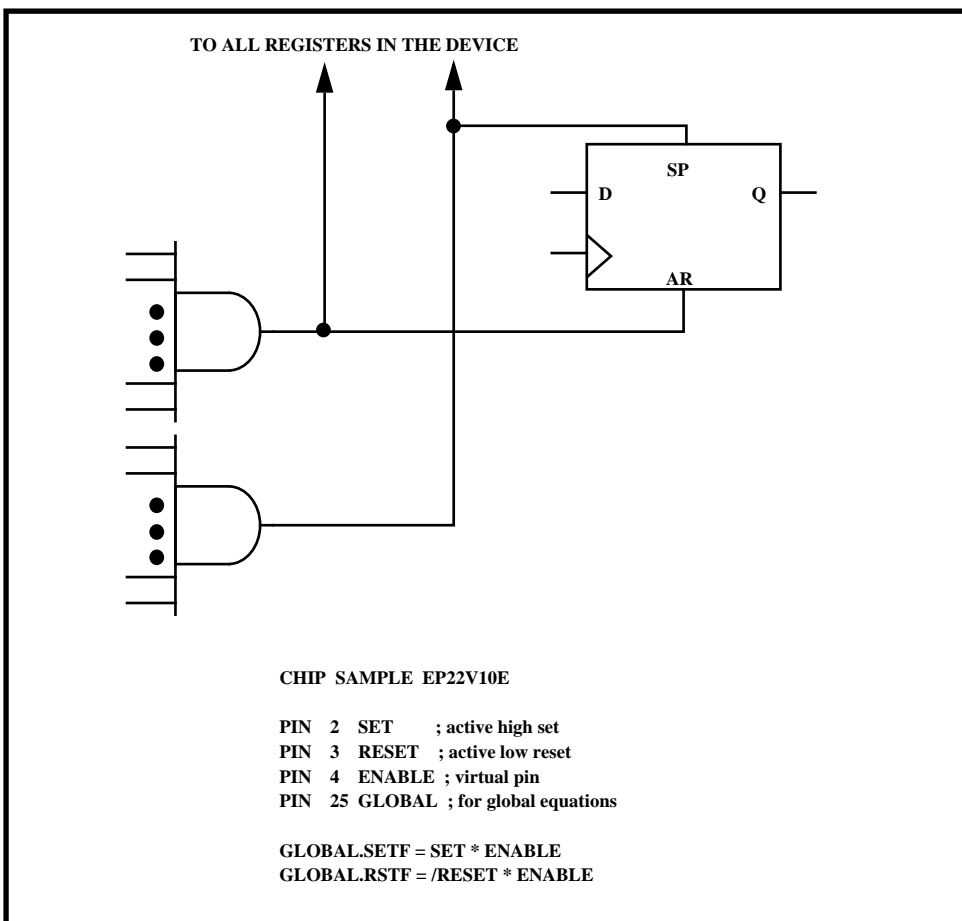


Figure 4-7 Global Set/Reset

Implementing Alternate I/O Options

The default I/O configurations for supported PLDs are shown in Figure 4-8. For combinatorial outputs, the default feedback option is “pin feedback.” This means that the feedback connection to the logic array comes from the I/O pin (after the output buffer). For registered outputs, the default feedback option is “registered feedback.” This means that the feedback connection to the logic array comes from the register output (before the output buffer). These defaults do not need to be specified in the PLDasm file (although you may desire to do so as part of your design methodology).

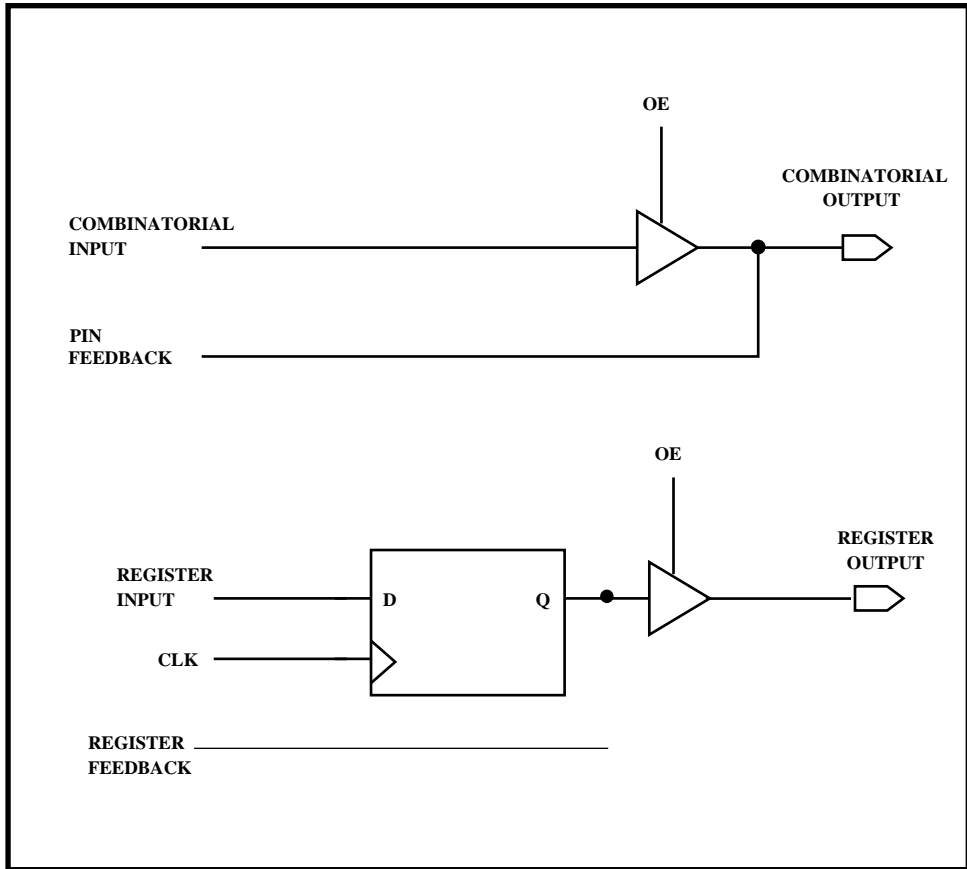


Figure 4-8 Default Feedbacks for Basic Output Types

Some Altera PLDs provide I/O configuration options in addition to those just described. For example:

- Many devices allow registers to use pin feedback instead of register feedback.
- Some devices allow combinatorial output with registered feedback.
- Global macrocells in many devices provide dual feedback paths to allow macrocells to

feed back their signals to the logic array while the I/O pin is still available for use.

To specify an alternate I/O configuration, use the appropriate keyword(s) in the pin declaration for I/O signals to be configured (see Figure 4-9). The keywords are not order-dependent, but you may wish to adhere to an order as part of your design methodology. If a keyword and the equation operator for an output conflict, the keyword takes precedence.

A complete list of all I/O keywords follows:

| I/O Keyword | Description |
|-----------------------|------------------------|
| 3VOLT | 3-Volt Drive Level |
| 5VOLT | 5-Volt Drive Level |
| CMOS_LEVEL | CMOS Voltage Level |
| COMBINATORIAL or COMB | Combinatorial Output |
| REGISTERED or REG | Registered Output |
| LATCHED | Latched Output |
| PINFBK | Pin Feedback |
| CMBFBK | Combinatorial Feedback |
| REGFBK | Registered Feedback |
| HIGH | Active-High Output |
| LOW | Active-Low Output |
| INPUT | Input Pin |
| OPEN_DRAIN | Open Drain Output |
| OUTPUT | Output Pin |
| I/O | I/O Pin |
| DELAYCLK | Delayed Clock |
| RAM | RAM Block |
| TTL_LEVEL | TTL Voltage Level |

```

; Default specifications - No keyword. Will be combina-
; torial or registered as determined by equation operator.

    PIN      5    COUT1
    PIN      6    ROUT1

; Combinatorial Output, Pin Feedback. Same as combinatorial
; default, but this time it is specified using keywords.

    PIN      5    COUT1    COMBINATORIAL    PINFBK

; Registered Output, Registered Feedback. Same as
; registered default, but this time it is specified
; using keywords.

    PIN      6    ROUT1    REGISTERED    REGFBK

; Registered Output, Pin Feedback.

    PIN      7    ROUT2    REGISTERED    PINFBK

; Combinatorial Output, Registered Feedback.

    PIN      15   CORF3    COMB    REGFBK

```

Figure 4-9 Examples of I/O Options Using PLDasm Syntax

The following pages show how to take advantage of these architectural features in PLDasm syntax. Note that the INPUT, OUTPUT, and I/O keywords are supported for compatibility with PALASM-2 source files only. The PLDasm compiler accepts these keywords but does not process files differently than when they are omitted.

Bi-directional I/O

Bi-directional I/O can be implemented on all supported Altera devices that provide pin feedback. The output enable to these pins determine whether the pin is an output or an input (see Figure 4-10). When the XMT_RCV p-term is active (high), the macrocell drives both the output pin and the input feedback signal. When XMT_RCV is low, any external device can drive the input feedback signal; the I/O pin acts as an input. Note that it is not necessary to list the bi-directional input signal in the input pin declaration to use this feature.

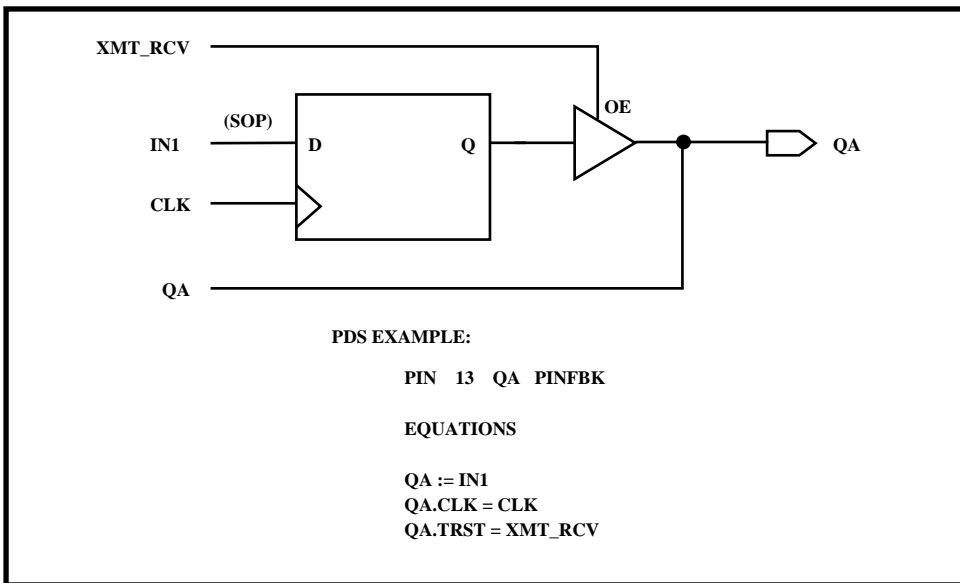


Figure 4-10 Bidirectional I/O Example

Dual Feedback Features

Some PLDs, such as the EPX8160, EPX780 and EPX740, contain dual-feedback paths. Dual feedback allows the associated I/O pin of a macrocell to be used for input or output (pin feedback). As shown in Figure 4-11, a second (internal) feedback can be taken from the output of a register, and the OE can be used as an input¹/output control pin. The following feedback support is provided:

- Support for dual feedback on *unassigned pins*.
- Pin feedback can be used as both an *input and an output*.

Unassigned Pins

The first benefit of the dual feedback feature is the ability to use unassigned pins to specify the feedback path. In order to maintain backwards compatibility with older designs (i.e., designs created with versions of PLDshell prior to V4.0), pin assignments can still be used to implement the dual feedback feature.

The PLDasm language feature developed to support dual feedback is the .IO language extension. This extension works with the .FB extension to permit a second feedback path to be specified without assigning a pin.

1. To the next macrocell.

Here's how they work:

- `.IO` *always* specifies the pin feedback path to be used as an input to another macrocell.
- `.FB` is defined by the feedback attribute in the PIN or NODE declaration section.

It is possible to use the `.IO` extension and there is no need to make a pin assignment. Figure 4-11 demonstrates the way to use `.FB` and `.IO`. In order to provide backwards compatibility with versions of PLDshell prior to V4.0, `.FB` and `.IO` mean the same thing for pin feedback.

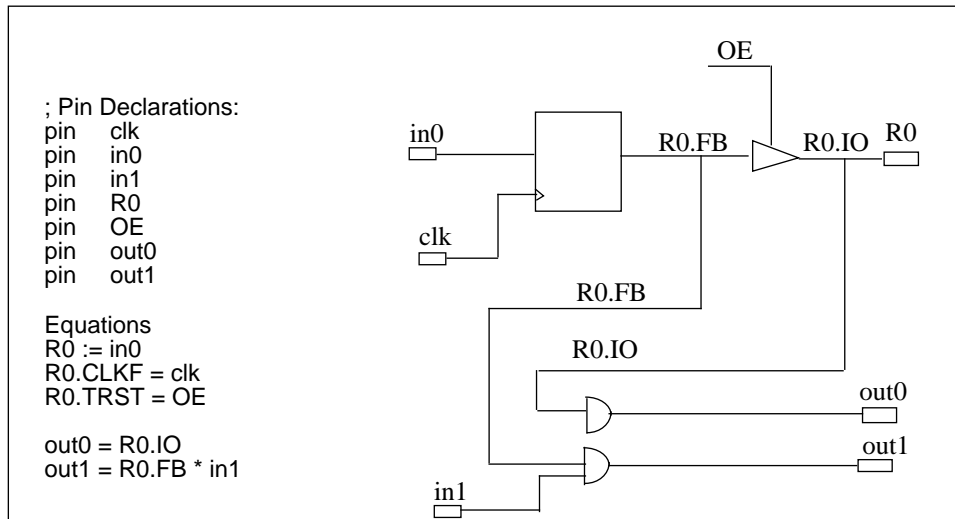


Figure 4-11 Dual Feedback Example

Default Feedback

The default feedback attributes for each macrocell are determined by the PIN or NODE declaration of the macrocell. Use of the `.IO` extension in an equation does not change the default feedback for the signal.

Default Extensions

If no extension is specified and feedback is being used as an input to an equation, the extension will default to `.FB`. The feedback could be either internal or pin feedback, depending on the specification.

Bi-Directional Feedback

The second benefit of the dual feedback feature is the ability to make dual feedback bi-directional. In versions of PLDshell prior to v4.0, pin feedback could only be used as an input. Now it can be used either as an input or an output, as shown in Figure 4-11.

Simple Dual Feedback Naming

The third benefit of the enhancement to dual feedback is that it offers a simpler use of the PLDasm language for feedback designs. In Figure 4-11, the .IO extension links both feedback paths to the same pin, permitting the pin names to better document the design.

Options

- Remember, there are two default feedback paths for the occasions when you do not specify a path. The defaults are as follows:

| <u>For this type of output,</u> | <u>...the default feedback type is:</u> |
|--|--|
| PIN A REG | registered feedback (REGFBK) |
| PIN A COMB | pin feedback (PINFBK) |

- The older method of creating dual feedback by specifying an input signal and an output signal on an assigned pin will still be supported, in addition to the .IO method.
- Both .IO and .FB are accepted in the EQUATION and SIMULATION sections of a design.

Limitations and Error Conditions

- The .IO extension works only on Altera FLEXlogic devices. It does not work on Classic devices. If the .IO extension is used in designs targeted to Classic devices, a fitter error will be generated.

Examples

Figures 4-12 through 4-15 show further examples of the use of the .IO extension in dual feedback.

- Figure 4-12 shows how PIN A takes on the value of the combinatorial feedback path of DFEED, while PIN B takes on the value of the pin feedback path for DFEED.
- Figure 4-13 shows PIN A taking on the value of the register feedback path of DFEED, while PIN B takes on the value of the pin feedback path for DFEED. This is the default configuration for a registered output.
- Figure 4-14 shows Output A and B taking on the value of the pin feedback path for DFEED. *This is the default configuration for a combinatorial output.*
- Figure 4-15 shows how PIN A takes on the value of the combinatorial feedback path of DFEED through a register, while PIN B takes the value of the pin feedback path of DFEED.

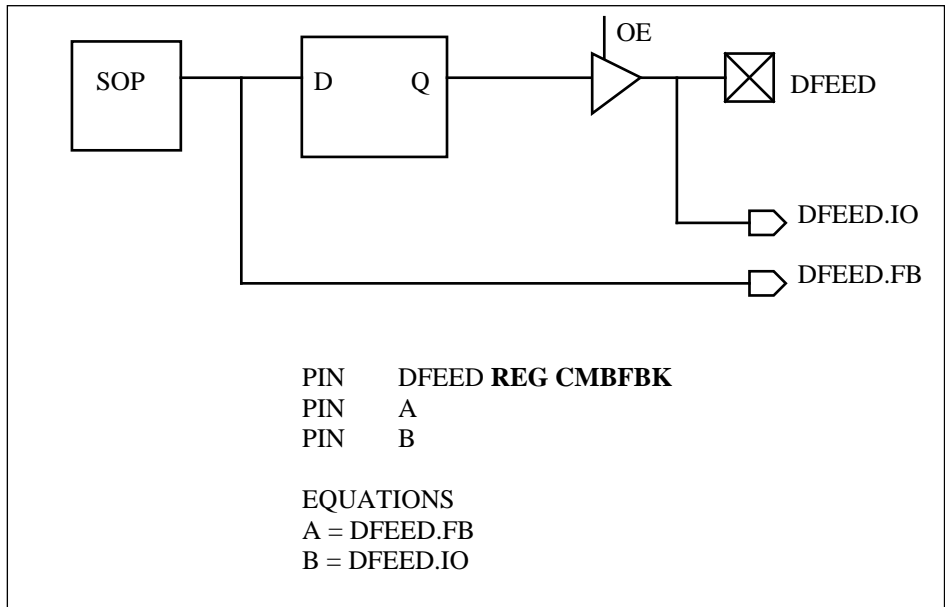


Figure 4-12 Combinatorial Dual Feedback

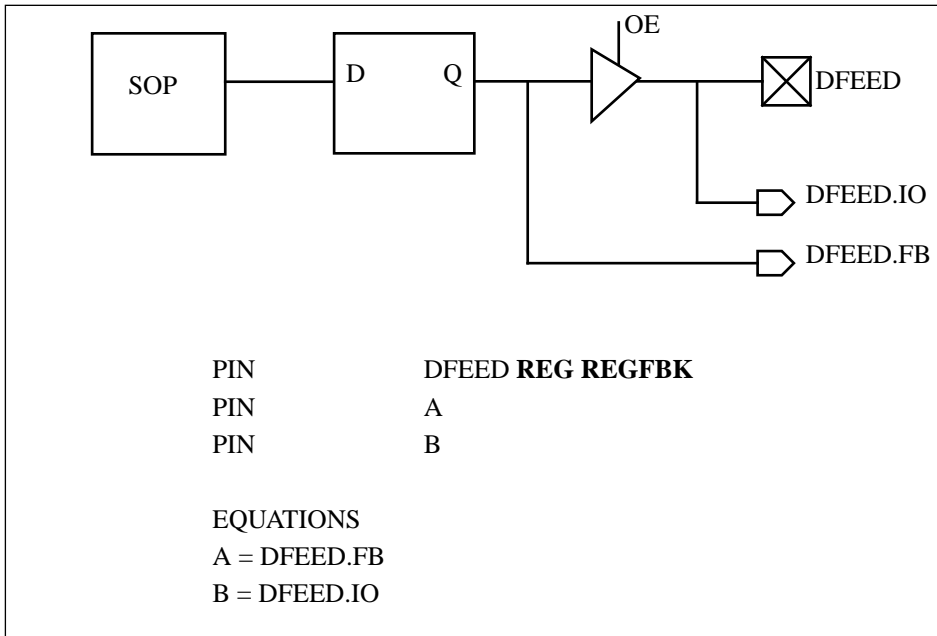


Figure 4-13 Registered Dual Feedback

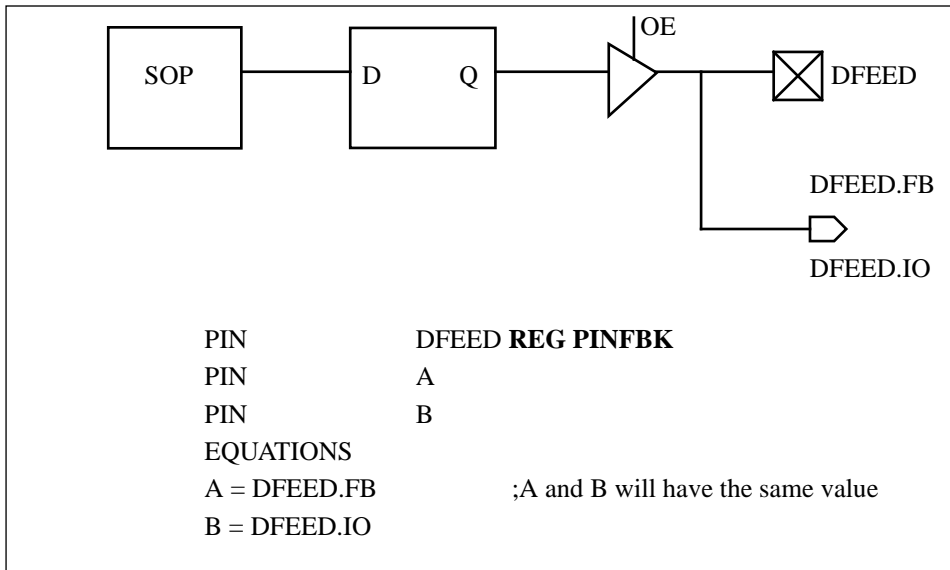


Figure 4-14 Pin Feedback

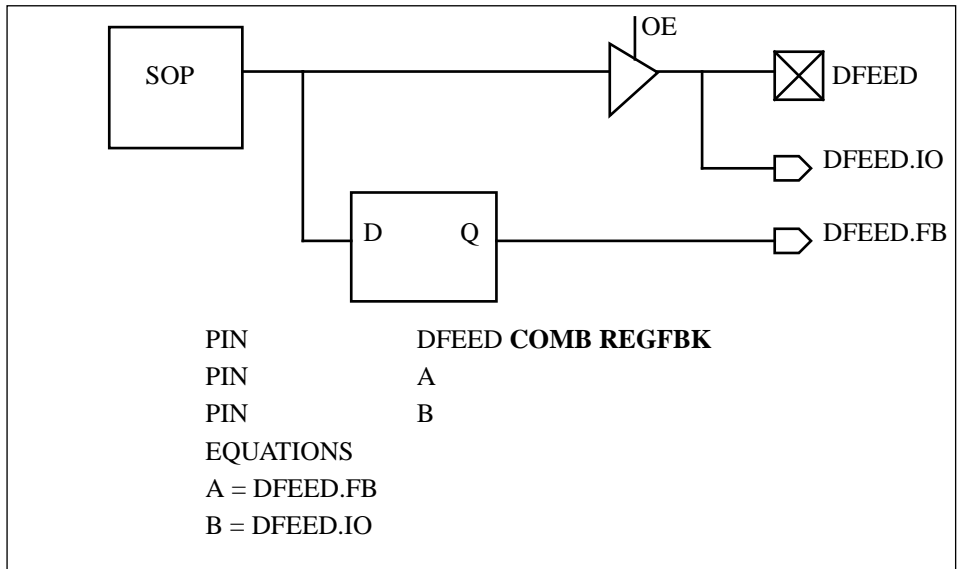


Figure 4-15 Combinatorial Output with Registered Feedback

P-Term Allocation

Some of the Altera PLDs supported by PLDshell allow p-terms from one macrocell to be allocated to other macrocells. This can give you from zero to 16 p-terms for your equations. Figure 4-16, P-Term Allocation on an EP312, shows an example where Macrocell 3 uses 16 p-terms, Macrocell 4 uses 4 p-terms, and Macrocell 5 uses 12 p-terms.

P-term allocation is performed automatically by the fitter. Designers take advantage of this feature by simply creating equations that meet their design requirements. The minimizer will reduce the number of p-terms and the fitter will generate the correct JEDEC image to allocate the needed p-terms.

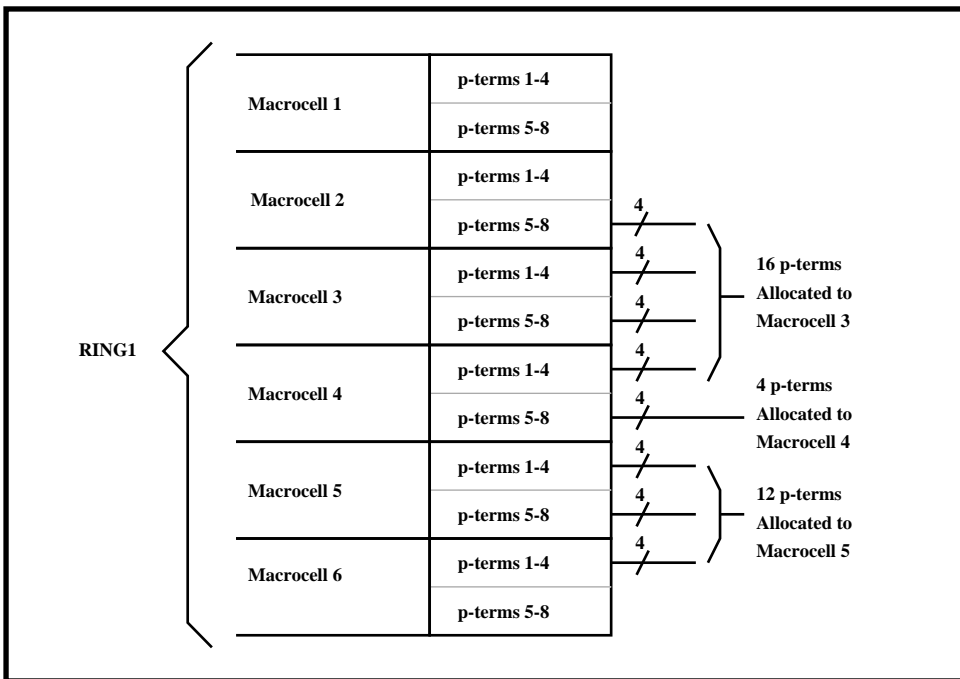


Figure 4-16 P-Term Allocation on an EP312

Using Disconnected Macrocells

In devices with p-term allocation, inputs to a macrocell are disconnected from the logic array when both p-term groups are allocated to adjacent macrocells. If all of the inputs to a macrocell are disconnected, the macrocell input can be tied to either VCC or GND (See Figure 4-17).

Also, the control signals, Output Enable, Preset, Clear, and Synchronous or Asynchronous Clock, are still available, which allows the macrocell to be used to implement a variety of useful functions, such as:

- Toggles using T-register and D-register equations
- Asynchronous RS registers driven by preset and clear p-terms
- Other registered functions that use the control signals

The ability to use disconnected registers can also be combined with feedback and dual-feedback support to make the most efficient use of device resources.

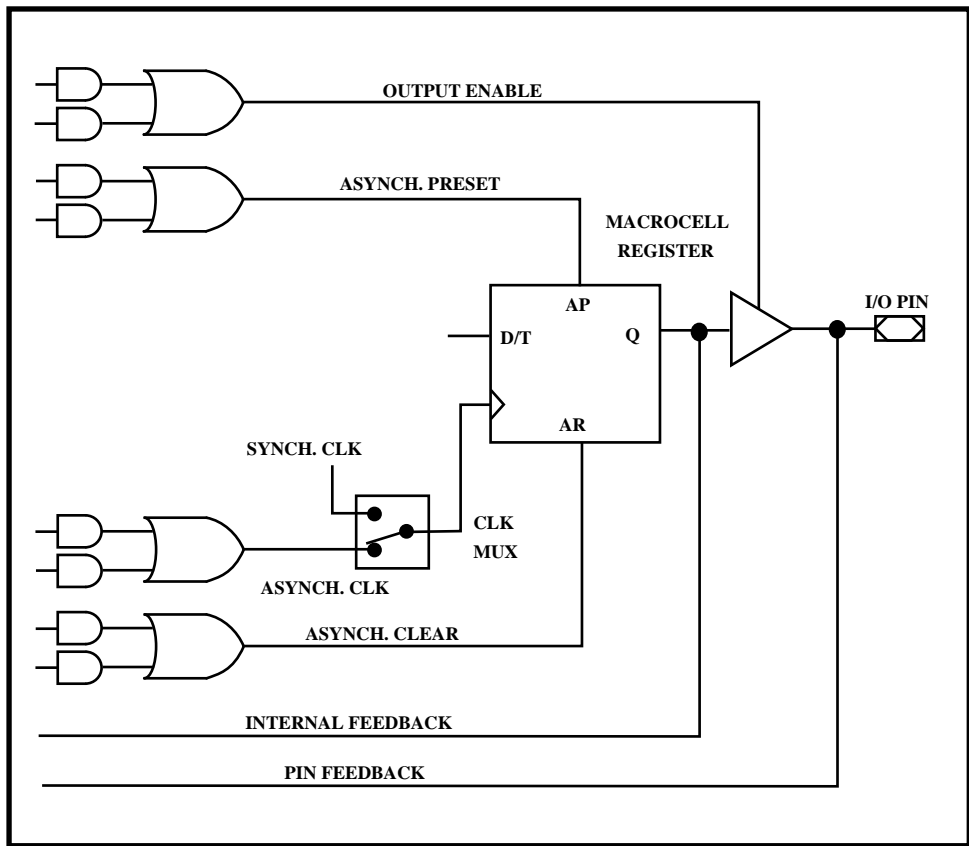


Figure 4-17 Using Disconnected Macrocells

Using Programmable Inputs

The EP312 and EP324 have programmable inputs that can be individually configured in one of five modes:

- Flow-through input
- Input register (D-register), synchronous operation
- Input register (D-register), asynchronous operation
- Input latch (transparent D-latch), synchronous operation
- Input latch (transparent D-latch), asynchronous operation

Figure 4-18, Programmable Input Examples, shows an example of a flow-through, a synchronous latch, and an asynchronous register. IN1 is a standard flow-through input. LIN2 is a latched input as specified by the LATCHED keyword in the pin assignment. The latch enable for LIN2 is specified as ILE in the equations section by using the signal name with a .LE extension. RIN3 is a registered input as specified by the REGISTERED

keyword in the pin assignment. This input register is asynchronously clocked by the invert of IN1. The RIN3 clock is specified as an asynchronous clock by using the .ACLK extension. When clocking latches/registers asynchronously (i.e., using a p-term), the clock can be generated by any p-term that can be expressed in a single AND form.

When using the synchronous ILE/ICLK clock pin, input latches open (become transparent) on the logic high and latch data on the falling edge of the signal. Input registers clock on the falling edge of the clock signal. (Note that this is different from macrocell registers, which clock on the rising edge of the clock signal.)

The ability to latch/clock data into the device inputs can help implement fast pipelining circuits.

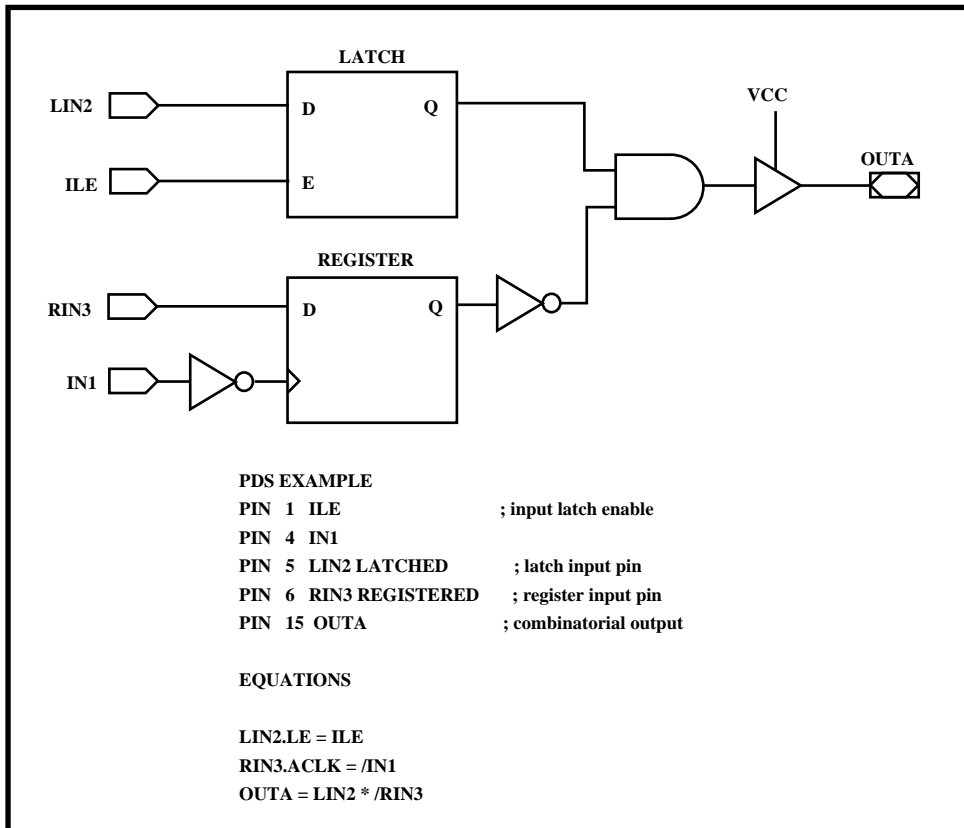


Figure 4-18 Programmable Input Examples

Altering Set-Up and Hold Times

On some of the Altera PLDs supported by PLDshell, there is a way to alter the t_{SU} (setup) and t_{CO} (clock-to-output) parameters to better match system timing. By using the asynchronous clocking feature provided with these devices, the t_{SU} and t_{CO} values are shifted with respect to the signal or signals clocking the registers. For example, using the asynchronous clock on the EP610 shortens t_{SU} and lengthens t_{CO} . Please refer to individual device data sheets for information on timing for Altera devices.

The following are PLDasm examples of synchronous and asynchronous clocking which will work for the EP610, EP600, EP910, and EP900:

Synchronous Clocking

```
PIN 1 CLK ; synch. 610/910 clock
PIN 10 OUT_S

EQUATIONS

OUT_S.CLKF = CLK
```

Asynchronous Clocking

```
PIN 2 ASYNCLK ; any input or I/O pin
PIN 10 OUT_S

EQUATIONS

OUT_S.ACLK = ASYNCLK
```

An alternate method of tailoring t_{SU} to system timing is provided on the EP22V10E in the form of a clock inversion option. Use of this feature allows user-selected registers to clock on the falling edge of a synchronous clock signal, effectively moving the t_{SU} windows with respect to the rest of the system timing. Refer to the discussion of “Registered Circuits” earlier in this chapter for an example.

Other devices, such as the EPX8160, EPX780 and EPX740, provide a delayed synchronous clock in addition to the asynchronous clock. This provides an additional t_{SU} and t_{CO} option. Refer to the corresponding data sheets for details.

Hardware Compare

The EPX8160, EPX780, and EPX740 provide a hardware compare option for each Configurable Function Block (CFB). Figure 4-19 shows an example of the hardware compare. This is an identity compare term that can feed the sum of products (SOP) expression of one of the outputs in a block. The compare term is not a sum of products function, and it will not be minimized by the compiler. A compare extension (.CMP) and double equal sign (==) have been added to the language to handle this feature.

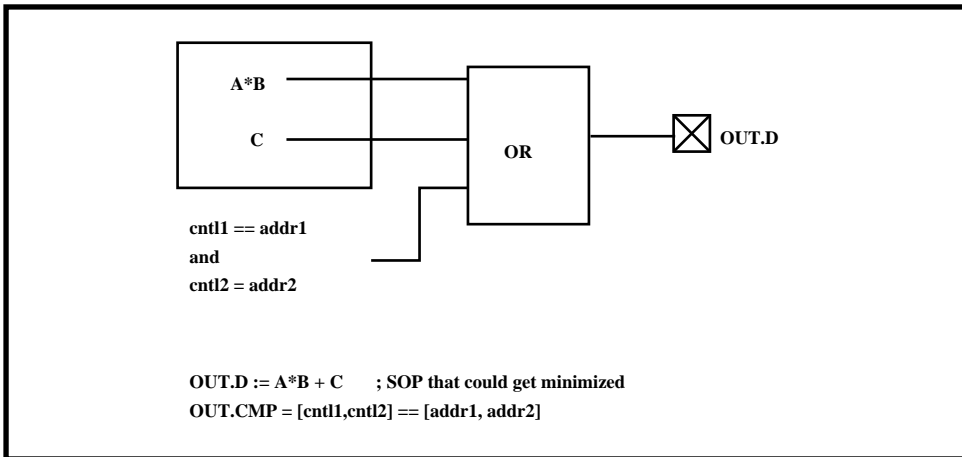


Figure 4-19 Hardware Compare Example

The result of the comparison (cntl1 vs. addr1 and cntl2 vs. addr2) would be OR'd into the rest of the SOP expression for a macrocell. If a compare is the only function needed, then the first line could be omitted or set OUT.D := GND.

3.3V/5V/Open Drain/Low Power

Some devices, such as the EPX8160, EPX780, and EPX740, allow outputs to swing to 3.3-volt or 5-volt levels. Keywords have been added to configure outputs and inputs to match application requirements. The following sections describe these keywords and show examples of their use.

Outputs

The specification of 3.3-volt versus 5-volt and standard versus Open Drain for output signals is handled in the PIN declaration keywords. New keywords have been added (3VOLT, 5VOLT, and OPEN_DRAIN) to specify 3.3-volt, 5-volt, and open drain outputs. The 3VOLT and 5VOLT keywords do not physically change the output levels. They allow the software to place these outputs in the same CFB. The VCCO pin on the CFBs are then tied to 3.3-volts or 5-volts.

The first line in Figure 4-20 defines Pin 12 as OUT1 and a 3.3-volt output.

The fourth line in Figure 4-20 is in the options section defining most or all outputs as 3.3-volts. This would mean that ALL outputs are 3.3-volt, unless explicitly changed on a pin declaration to be 5-volt (5VOLT). The software will group like voltage signals to the same CFB. The second line in the figure is a 5-volt output.

The OPEN_DRAIN keyword physically changes an output by disabling the internal pull-up. An external pull-up can then be used. The second line in Figure 4-20 shows an open drain example.

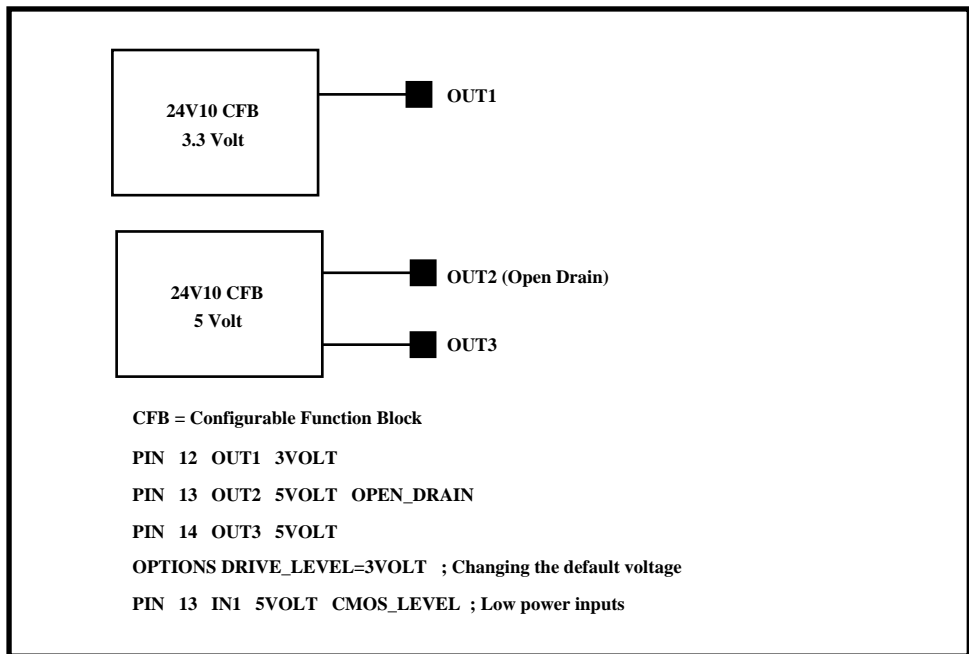


Figure 4-20 3.3V/5V/Open Drain/Low Power Example

Inputs

Two keywords allow inputs and I/O pins used as inputs to be configured for CMOS or TTL levels. `CMOS_LEVEL` is used for 5V CMOS inputs. `TTL_LEVEL` is used for 3V CMOS and TTL input levels. This allows designers to optimize applications for I_{SB} (standby current) or leakage current in some situations. For example:

```
PIN 13 IN1 5VOLT CMOS_LEVEL
OR
PIN 18 IN2 3VOLT TTL_LEVEL
```

The options section can be used to change the default input voltage if all or the majority of inputs are the same. For example:

```
OPTIONS
INPUT_PULLUP = CMOS_LEVEL
```

This defines all inputs for CMOS levels, unless explicitly declared to be TTL level. (The CMOS/TTL level keywords enable/disable internal pullups for inputs and I/O pins used as inputs.)

Hierarchy

Design hierarchy is supported for all devices with the addition of `DEFMOD`, `ENDMOD`, and `MODULE` constructs. Hierarchy with source files allows large designs to be developed

from smaller modules or source files. The following syntax is used to call a lower-level module:

```
MODULE <module name> [INSTANCE <instance name>] [FILE <file name>]
( <argument list> )
```

The <instance name> is used to distinguish internal (buried) signal names, especially when there are multiple module calls. The following example would cause the internal name BURIED to appear as A_BURIED in the high-level design:

```
MODULE mod1 INSTANCE A (...)
```

NOTE:

The instance names should be short because they will be combined at each level of module call, up to the maximum signal name length. Then, a unique number will replace the entire instance name prefix.

For <file name>, the .PDS extension should not be used. The <module name> can be the same as the <file name> or can be different. If the <module name> is the same as the <file name>, no file name is required. If the file <file name> is not specified, the module is first looked for in the current file. The module will then be looked for in the file with the name of the module.

The following syntax is used to define a lower-level module:

```
DEFMOD <module name> ( <argument list> )
.
.
ENDMOD
```

(The “. . .” refers to valid PDS syntax that defines a function.) The parser looks within the same file for module names or in the designated file for file names. This allows libraries of macros or modules to be developed and used as separate files or as modules within a single file. An example is included here. Refer to Chapter 7 for additional information.

3-Level Modular Example

The following is an example that includes three .PDS files and three levels of hierarchy.

The first module is a 2-bit XOR design. The module name is **2_bit_xor**; the file name is 2BXOR.PDS. This is the lowest level function for this example.

```
DEFMOD 2_bit_xor (in1, in2, out) ; referenced by 2BCMPR.PDS
CHIP mod Altera_arch
PIN in1
PIN in2
PIN out
EQUATIONS
    out1 = in1 * /in2 + /in1 * in2
ENDMOD
```

The second module is a 2-bit comparator. The module name is **2_bit_compare**; the file name is 2BCMPR.PDS. This module makes two calls to **2_bit_xor** in 2BXOR.PDS.

```

DEFMOD 2_bit_compare (a1, a2, b1, b2, cmp1, cmp2); referenced by
                                                ; HIGATE1.PDS

CHIP mod Altera_arch
PIN a1
PIN a2
PIN b1
PIN b2
PIN cmp1
PIN cmp2
MODULE 2_bit_xor FILE 2bxor(in1=a1, in2=b1, out=cmp1)
MODULE 2_bit_xor FILE 2bxor(in1=a2, in2=b2, out=cmp2)
ENDMOD

```

At the highest level for this example, HIGATE1.PDS calls **2_bit_compare** in 2BCMPR.PDS. Figure 4-21, Hierarchy Example, illustrates the hierarchy for this example.

```

CHIP mydesign EP22v10
PIN addr1
PIN addr2
PIN baddr1
PIN baddr2
PIN we1
PIN we2
MODULE 2_bit_compare FILE 2bcmpr (a1=addr1, a2=addr2, b1=baddr1,
                                   b2=baddr2, cmp1=we1, cmp2=we2)

```

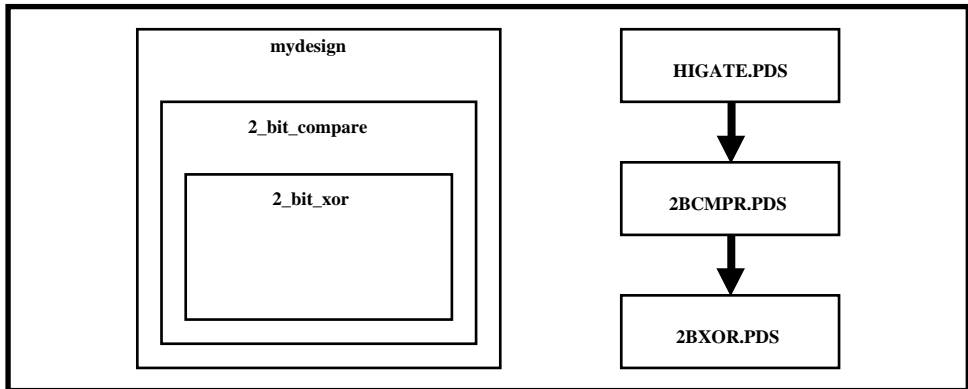


Figure 4-21 Hierarchy Example

HIGATE2.PDS shows how to use the same constructs in a single source file. In this case, the file references itself. This capability is similar to a software program in which a main loop calls subroutines in the same file. The design Merge utility provides a semi-automated method of using this syntax to create a higher-level .PDS file from multiple lower-level .PDS files.

NOTE:

The ability to independently specify file name and module

names allows collections of modules to be grouped into library files. Refer to Chapter 7, Modules and Hierarchical Design, for additional information and examples.

Vector Notation in Design Section

Vector/set notation is supported in the Declarations section of your .PDS source file, between STRING and FUSE statements and before the OPTIONS statement. For example:

```
OUT[0:3] := A[3:6] * B[0:3]
```

is equivalent to

```
OUT0 = A3 * B0  
OUT1 = A4 * B1  
OUT2 = A5 * B2  
OUT3 = A6 * B3
```

Groups of signals could be defined in the PIN declaration as follows:

```
PIN bus[0:31]
```

Or, they could be defined from existing signals as follows:

```
VECTOR addr := [addr6, addr5, addr4, addr3, addr2, addr1, addr0]
```

Ranges of sequentially numbered signals can be described similarly:

```
x[0:7]
```

is the same as

```
x0, x1, x2, x3, x4, x5, x6, x7
```

RAM

Each CFB in FLEXlogic devices can be configured as a block of 128x10 SRAM instead of a PLD block. An SRAM block may have up to ten outputs, three active-low² control inputs (block enable, write enable, and output enable), ten data inputs, and seven address control lines. Care must be taken when declaring SRAM control signals as nodes. In order for compilation to function properly, the *Automatic Inversion feature* in the Compile Options submenu must be turned off (i.e., set to NO). Then, when the compile is in progress you should answer no to auto-inversion only on the SRAM signals. Also, (S)RAM names should not end with a numeric character. Vectored signals will be processed incorrectly and the compile will abnormally terminate if this rule is not adhered to. Figure 4-22, Signal Grouping Example, is an example of the .PDS specification of a RAM block.

2. The EPX8160 offers the choice of active-high or active-low control inputs.

Additionally, individual outputs may be buried (either by the physical device, or by the user) by grounding their OEs. For example, the RAM bits 5 and 7-9 could be buried using the following:

```
BUFRAM[5].TRST = GND
BUFRAM[7:9].TRST = GND
/BUFRAM.WE = GND
```

RAM Defaults

A RAM block can be initialized with a RAM_DEFAULTS section in the .PDS file. It is important to note that RAM default values can only be made from most significant bit to least significant bit. The advantage of a RAM defaults section is to set JEDEC bits so that at device start-up the memory would be initialized to the specified values. For example:

```
RAM_DEFAULTS myrom
;address value
0 : 0xF
[0xF:0x1] : 0x0
[0x7F:0x10] : 0x3FF
```

This example would set the first 10-bit word to the value of 15, the next 15 words to the value of 0, and the rest of the bits to all 1s.

By tying the Write Enable to low or GND, the SRAM block becomes a ROM.

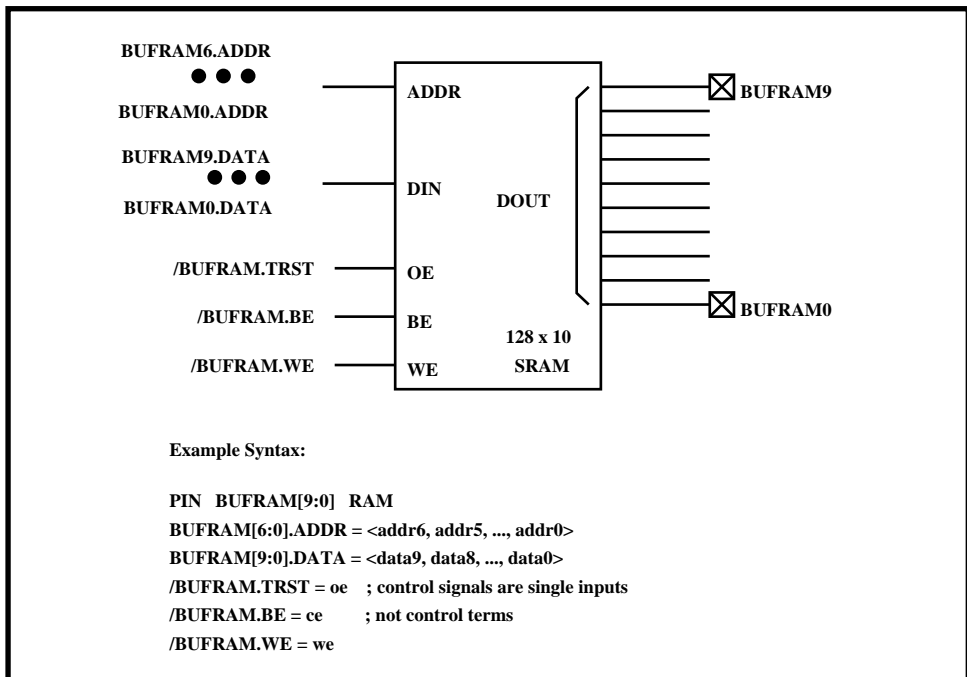


Figure 4-22 Signal Grouping Example

The keyword `DEFAULT_VALUE`, followed by a value, can be used to initialize the whole SRAM block to the same value, as follows:

```
RAM_DEFAULTS BUFRAM DEFAULT_VALUE 0x155
```

This will initialize all locations of the SRAM to the value of 155H.

Delayed Clock

There are three clocking modes on the EPX8160, EPX780, and EPX740 devices:

- Synchronous — driven directly from a dedicated clock pin.
- Synchronous with delay — provides a slight (~ 2 ns.) delay between the clock pin and macrocell to shift the t_{SU} , t_{CO} window with respect to the clock.
- Asynchronous — via a p-term in the logic array; this provides yet another shift of t_{SU} and t_{CO} with respect to the clock.

The delayed synchronous mode is supported with the addition of a `DELAYCLK` pin declaration keyword to be applied to outputs that should use the delayed clock. This keyword is used with the following syntax:

```
PIN out1 DELAYCLK
```

There are two synchronous clocks feeding each block in the device. A delayed clock is applied on a block-by-block basis, and thus becomes another grouping criteria (e.g., 3.3VOLT/5VOLT, RAM). For example:

```
PIN 1 clk ; synchronous clock
PIN out1 DELAYCLK
PIN out2 DELAYCLK
PIN out3
...
out1.CLKF = clk;uses delayed clk
out2.CLKF = clk;uses delayed clk
out3.CLKF = clk;uses standard clk
```

Buried Macrocells

Buried macrocells can be specified for all devices by use of the `NODE` keyword in place of a PIN declaration. For example:

```
NODE 23 x_buried
```

is equivalent to

```
PIN 23 x_buried
...
x_buried.TRST = GND
```

User Electronic Signature (UES) Bits

EPX8160, EPX780, and EPX740 provide storage for a user electronic signature pattern. Support for the user electronic signature bits (UES) is provided by using the SIGNATURE statement. Below is an example of SIGNATURE keyword placement in a PDS file.

```
CHIP Mychip Altera_Arch
SIGNATURE = 0xffae2
PIN      1      clk
```

The SIGNATURE bits can be read out of the JTAG port using the UESCODE instruction.

I/O, Feedback Macrocell Combinations

The EPX8160, EPX780, and EPX740 devices also allow for combinatorial/register feedback independent of the output type. This means that combinatorial output is possible with registered feedback. Also, registered feedback is possible with combinatorial output. These architectural combinations are supported with a pin declaration keyword, REGFBK. For example, in the following T flip-flop:

```
PIN   tout      CMBFBK
PIN   cout_t    TREGFBK
...
tout.T := ...
cout_t = ...
```

defines

```
tout.T as T output, combinatorial feedback
cout_t as combinatorial output, T register feedback
```

Truth Table Design

Truth tables provide an effective way of describing a design or parts of a design. For example, truth tables are often used for decoders. PLDasm supports more than one optional Truth Table section in PLDasm files.

As shown in Figure 4-23, Truth Table Examples, each Truth Table section begins with the T_TAB keyword. Truth tables are position dependent, i.e., each input has a corresponding column for each row of the table. The first line of the table lists the input and output signal names for the truth table inside parentheses. Subsequent rows list the values of each output for each combination of inputs.

```

; Combinatorial truth table

T_TAB (in1 in2 >> out1 out2 /out3)
0 0 : 0 0 0
0 1 : 0 0 1
1 0 : 1 0 0
1 1 : 1 1 0

; Registered truth table

T_TAB(UPDOWN /CLR q0 q1 :>> q0.T q1.D)
1 0 0 1 : 0 0
1 0 1 0 : 0 0
0 0 0 1 : 1 1
0 0 1 1 : 1 0
X 0 0 x : 1 1

```

Figure 4-23 Truth Table Examples

Inputs are listed first. These are followed by a separator. Outputs are then listed. Entries can be separated by blank spaces or commas. The separator in the signal name line also identifies truth table outputs as combinatorial (>>), registered (:>>), or latched (*>>). The sample file ADDR1.PDS in your installation directory shows a truth table with latched outputs.

All outputs in the same truth table must be the same type (i.e., you cannot mix combinatorial and registered outputs in the same truth table). The first table in Figure 4-23 is combinatorial; the second is registered. The separator for all other lines in the table is a colon (:).

Legal values for truth table entries are as follows:

```

0 = False
1 = True
X = Don't Care

```

Don't Cares may be used to describe input or output values, but a Don't Care for an output is treated as a logical 0.

Inputs or outputs can be specified as true or complemented. For example, out2 is true and /out3 is complemented. Thus /out3 goes high for cases where the truth table entry specifies a 0; it goes low where the truth table entry specifies a 1.

For truth tables using registered outputs, the register type can be selected by adding the appropriate extension to the output signal name. In the second truth table, q0.T specifies a T-type register and q1.D specifies a D-type register. This feature can be useful when it is known that a particular type will end up using fewer p-terms, but only when using devices that support T-type registers.

Truth table outputs can only drive output pins directly. They can be used as inputs to Boolean equations, state machines, or other truth tables indirectly, i.e., via feedback paths to the logic array. All truth table outputs must be specified in the pin list.

Truth table inputs can come from input or I/O pins. This means that Boolean equations, state machines, or other truth tables can provide input to truth tables via feedback paths to the logic array.

A Truth Table section ends with the next PLDasm keyword.

State Machine Design

State machines provide an effective way of describing sequential (registered) logic. A designer typically draws a state diagram that represents the different states and transitions for a design. This diagram can then be expressed in PLDasm's state machine syntax. PLDasm supports more than one optional State Machine section in PLDasm files.

Each state machine begins with the STATE keyword followed by a machine type keyword (MEALY_MACHINE or MOORE_MACHINE). Outputs on Moore machines depend on the current state only. Outputs on Mealy machines depend on both the current state and next state information. The machine type keyword is followed by subsections that identify the global defaults, state transitions, output values, and transition conditions. Synchronous state machines transition on the rising edge of a dedicated clock pin. Asynchronous state machines transition when the specified condition used as the clock is true.

NOTES:

These are not “Next-State” tables. The “1s” in the output (right) side indicate input p-terms to take from the input (left) side of the table.

PLDasm supports asynchronous state machines, but does not perform hazard checking. You should check the resulting equations for hazard conditions to ensure that your design will function properly.

Simple Moore State Machine Example

Figure 4-24, 2-Bit Counter State Machine, is a sample State Machine section that implements a 2-bit up/down counter (2BIT.PDS in your installation directory). The counter is a Moore machine defined to have four states (S1 through S4), with state S1 as the default state. The state assignments subsection defines the output values for each state. As can be seen, the outputs follow a binary sequence. The state transitions subsection defines the order of transitions from one state to another based on transition conditions.

```

CHIP 2_bit_count EP220

; pins

PIN 1   CLK
PIN 2   UPDOWN
PIN 3   CLEAR
PIN 12  Q1
PIN 13  Q0

; Simple 2-Bit State Machine

STATE
MOORE_MACHINE
DEFAULT_BRANCH S1

; state assignments

S1 = /Q1 * /Q0
S2 = /Q1 * Q0
S3 = Q1 * /Q0
S4 = Q1 * Q0

; state transitions

S1 := UP  -> S2
    + DOWN-> S4

S2 := UP  -> S3
    + DOWN-> S1

S3 := UP  -> S4
    + DOWN-> S2

S4 := UP  -> S1
    + DOWN-> S3

; input conditions transitions

CONDITIONS

UP = UPDOWN * /CLEAR
DOWN = /UPDOWN * /CLEAR

```

Figure 4-24 2-Bit Counter State Machine

The conditions subsection is denoted by the CONDITIONS keyword. In the figure, UP and DOWN are mutually exclusive, with both qualified by CLEAR. This means that when CLEAR is asserted (high), the default branch to S1 is taken, which clears the counter to 00. When CLEAR is not asserted (low) the state machine counts up when UPDOWN is high and counts down when UPDOWN is low. Input conditions can only be combinatorial equations. They can be specified in the Conditions subsection or in the transitions section. When specified in the transitions section, they should be enclosed in parentheses, as follows:

```
S7 := BUS_CONT-> S8 ; S8 if ready
+ (/RDY * RESET * B2) -> S10 ; S10 if not ready
```

State Machine Format (Moore Machine)

This section of the guide describes state machine format for PLDasm files in greater detail. In Figure 4-25, the STATE keyword and the machine type keyword are followed by five subsections: (1) Machine Defaults, (2) State Assignments, (3) State Transitions, (4) Transition Outputs, and (5) Transition Conditions. (A Mealy State Machine appears later in this section.)

Machine Defaults

The Machine Defaults subsection can include three defaults: Output Hold, Default Output, and Default Branch. Each State Machine section can use one, two, or all three defaults. When using more than one default, they must appear in the order shown.

The OUTPUT_HOLD keyword, followed by a list of output pins, specifies that those outputs are to hold their previous state if it is not possible to determine which state to transition to. Use of this option prevents outputs from changing states when all possible state transitions are not specified. Figure 4-25 uses this default.

The DEFAULT_OUTPUT keyword, followed by a list of output pin names, defines the default levels for those outputs if it is not possible to resolve a transition. Use of this option prevents outputs from going to unknown states when all possible state transitions are not specified.

```

Title      Local Bus Controller Example
Pattern    pds
Revision   1
Author     Your Name
Company    Your Company
Date       Date

CHIP BUS_CON1 EP224

; inputs
PIN 1      CLK
PIN 2      M_IO
PIN 3      INT_CYC
PIN 4      A20
PIN 5      READY

; outputs
PIN 15     LOCAL
PIN 16     MEMORY
PIN 17     INTACK
PIN 18     Q3
PIN 19     Q2
PIN 20     Q1
PIN 21     Q0
; simple bus controller
; inputs are M_IO, INT_CYC, A20, and READY
; outputs are LOCAL, MEMORY, and INTACK
STATE
MOORE_MACHINE
; define defaults

OUTPUT_HOLD LOCAL MEMORY INTACK

DEFAULT_BRANCH S0
; state assignments

S0 = /Q3 * /Q2 * /Q1 * /Q0
S1 = /Q3 * /Q2 * /Q1 * Q0
S2 = /Q3 * /Q2 * Q1 * /Q0
S3 = /Q3 * /Q2 * Q1 * Q0
S4 = /Q3 * Q2 * /Q1 * /Q0
S5 = /Q3 * Q2 * /Q1 * Q0
S6 = /Q3 * Q2 * Q1 * /Q0
S7 = /Q3 * Q2 * Q1 * Q0
S8 = Q3 * /Q2 * /Q1 * /Q0

```

Figure 4-25 Example State Machine Section

```

; state transitions

S0 := MEM_REQ-> S1; S1 on local memory request
+ INT_REQ-> S3; S3 on interrupt request
      ; no-op (S0) if no request

; memory cycles

S1 := VCC -> S2      ; one fixed wait cycle

S2 := BUS_CONT-> S7; S7 if ready
+ BUS_WAIT-> S2; stay if not ready
; interrupt cycles

S3 := BUS_CONT-> S4; S4 if ready
+ BUS_WAIT-> S2; stay if not ready

S4 := VCC -> S5      ; jump to S5 - fixed wait
S5 := VCC -> S6      ; jump to S6 - fixed wait

S6 := BUS_CONT-> S7; jump to S7 if interrupt done
+ BUS_WAIT-> S6; stay if not done

; cleanup and idle cycles

S7 := VCC -> S8      ; cleanup, then jump to S8
S8 := VCC -> S0      ; jump to S0, ready to start

; transition outputs

S0.OUTF := LOCAL * /MEMORY * /INTACK
S1.OUTF := /LOCAL * MEMORY * /INTACK
S2.OUTF := /LOCAL * MEMORY * /INTACK
S3.OUTF := /LOCAL * /MEMORY * INTACK

S4.OUTF := /LOCAL * /MEMORY * /INTACK
S5.OUTF := /LOCAL * /MEMORY * /INTACK
S6.OUTF := /LOCAL * /MEMORY * INTACK

S7.OUTF := /LOCAL * /MEMORY * /INTACK
S8.OUTF := LOCAL * /MEMORY * /INTACK

CONDITIONS

MEM_REQ = M_IO * /INT_CYC * /A20
INT_REQ = /M_IO * INT_CYC * /A20

BUS_CONT = /READY
BUS_WAIT = READY

```

Figure 4-25 Example State Machine Section (Continued)

The `DEFAULT_BRANCH` keyword, followed by the name of a state or additional keyword, defines the default state to which the machine will transition if it is not possible to decode input conditions. Use of this option eliminates the need to specify all possible combinations of input conditions. Should an unspecified input combination occur, the machine will behave as specified. Three behaviors are possible:

- Transition to the default state. This is specified by including the name of the default state after the keyword, as follows:

```
DEFAULT_BRANCH S1
```

- Hold the current state. This is specified by including the keyword `HOLD_STATE` after the `DEFAULT_BRANCH` keyword, as follows:

```
DEFAULT_BRANCH HOLD_STATE
```

- Transition to the next state. This is specified by including the `NEXT_STATE` keyword after the `DEFAULT_BRANCH` keyword. The next state is defined by the order of the state assignment equations. This option is implemented as follows:

```
DEFAULT_BRANCH NEXT_STATE
```

State Assignments

The State Assignments subsection defines each of the states in the machine, the state variables, and the values of those variables. States are defined by a state name, i.e., `S1`, `S2`, etc., on the left-hand side of an equal sign (=). State variables and the values of those variables are listed on the right-hand side of the equal sign. Each state must have a unique set of values for the state variables. When defining values for state variables, the choices are true (no prefix) or false (/ prefix).

State variables can have from 1 to 14 alphanumeric characters or underscores; the first character must be an alphabetic character.

Three possible methods for state assignment may be used:

Binary State Assignment

Binary is the simplest state assignment method. Each state in the list of state assignments is encoded with the binary representation. The state machine shown below shows an example of binary state assignment. This is also the method used in Figure 4-25.

```

STATE
MOORE_MACHINE
DEFAULT_BRANCH S1

; state assignments
S1 = /Q1 * /Q0
S2 = /Q1 * Q0
S3 = Q1 * /Q0
S4 = Q1 * Q0

```

Gray Code State Assignment

A Gray code is defined as two consecutive states that differ in state assignment values by exactly one bit. The example below shows assignment of Gray codes for an 8-state machine.

```

STATE
MOORE_MACHINE
DEFAULT_BRANCH S0

;state assignments
S0 = /Q2 * /Q1 * /Q0      ; state 0, powerup
S1 = /Q2 * /Q1 * Q0
S2 = /Q2 * Q1 * Q0
S3 = /Q2 * Q1 * /Q0
S4 = Q2 * Q1 * /Q0
S5 = Q2 * Q1 * Q0
S6 = Q2 * /Q1 * Q0
S7 = Q2 * /Q1 * /Q0      ; state 7

```

One Hot State Assignment

With a One Hot code, there are generally as many output bits as there are states in the machine. Each bit is 0 except for the state that the machine is in. This is primarily useful for very small machines, or for those where the state values are driving other logic. An example of One Hot assignment is shown below:

```

STATE
MEALY_MACHINE
DEFAULT_BRANCH S0

;state assignments
INITIAL = /Q3 * /Q2 * /Q1 * /Q0      ; state 0, powerup
S1 = /Q3 * /Q2 * /Q1 * Q0
S2 = /Q3 * /Q2 * Q1 * /Q0
S3 = /Q3 * Q2 * /Q1 * /Q0
S4 = Q3 * /Q2 * /Q1 * /Q0

```

State Transitions

The State Transitions subsection specifies the transitions between states. The subsection begins with the name of a state. State transitions that are not explicitly defined in this section will transition to the DEFAULT_BRANCH state, if any.

Each entry in this subsection contains the name of the current state, the Boolean register operator (:=) or the combinatorial operation (=), a condition label, the transition designator (->), and the name of the target state. For example, the first entry in the State Transitions subsection in Figure 4-25 is as follows:

```
S0 := MEM_REQ->S1; S1 on local memory request
    + INT_REQ->S3; S3 on interrupt request
      ; no-op (S0) if no request
```

This entry has the following meaning: “When you are at state S0 and MEM_REQ is true, transition to state S1 on the next clock edge; if INT_REQ is true, transition to state S3 on the next clock edge.” Since S0 was previously defined as the Default Branch state, if MEM_REQ or INT_REQ are not true, the machine would remain in state S0.

The second entry is as follows:

```
S1 := VCC->S2 ; on fixed wait cycle
```

Since no condition is specified, this is an *unconditional transition*. The machine will be in state S1 for one clock cycle before transitioning to state S2. If the Default Branch had been defined as NEXT_STATE, it would not have been necessary to specify each unconditional transition; they would be assumed.

The third entry is as follows:

```
S2 := BUS_CONT-> S4; S4 if ready
    + BUS_WAIT-> S2; stay if not ready
```

This entry has the following meaning: “When you are at state S2 and BUS_CONT is true, transition to state S4. If BUS_WAIT is true and BUS_CONT is not true, remain in state S2.” S0 is still the default state if neither BUS_CONT or BUS_WAIT is true; this case, however, will never occur because BUS_WAIT is defined in the Conditions section to be the complement of BUS_CONT. Conditions for a given state transition must be mutually exclusive, but do not have to be the complement of each other.

An *unconditional else* combines the “+” and “->” to provide the default transition for a given state. This unconditional else overrides the default branch defined for the machine, but only for the state specified. For example:

```
SA := READYNOW -> SB ;SB if ready now
    +-> SC           ;SC if not ready now
```

This entry has the following meaning: “when you are at state SA and READYNOW is true, transition to state SB. Else, transition to state SC.”

For asynchronous state machines, use the combinatorial operator (=) for state transitions, as follows:

```
S8 = WAIT_DONE-> S9; jump on wait_done high
```

NOTE:

PLDasm supports asynchronous state machines, but does not perform hazard checking. You should check the resulting equations for hazard conditions to ensure that your design will function properly.

Note that Boolean expressions can be used in the State Transition subsection itself, instead of the Transitions Conditions subsection. For example:

```
SD := ( A * /B ) -> SE
```

is the same as

```
SD := DONE -> SE
CONDITIONS
DONE = ( A * /B )
```

Transition Outputs

The Transition Outputs subsection specifies the state of output signals during transitions for state machines where outputs are not the state registers themselves. In this case, the output signals are LOCAL, MEMORY, and INTACK while the state registers are Q0 through Q3.

Note that LOCAL, MEMORY, and INTACK have previously been defined in the OUTPUT_HOLD specification. This means that they will hold their previous state if it is not possible to resolve a transition to the next state.

The S0.OUTF entry in Figure 4-25 has the following meaning: “If you are transitioning from S0, LOCAL is driven high and MEMORY and INTACK are driven low.”

The entry for S1.OUTF in Figure 4-25 has the following meaning: “If you are transitioning from S1, LOCAL and INTACK are driven low and MEMORY is driven high.

Another way to specify default output transitions is as follows:

```
P1.OUTF := CONDITION_1 -> /SIGNALA * SIGNALB
          + CONDITION_2 -> SIGNALA * /SIGNALB
          +--> /SIGNALA * /SIGNALB
```

where the “+>” says “transition the outputs as follows if the above conditions are not met.” This defines default output transitions for a particular state based on conditions. There may be only one such definition for each state. Note that the *fact that conditions are added to the output transitions* defines these outputs as Mealy outputs (refer to the next example). When no conditions are present, the outputs are Moore outputs.

Transition Conditions

The Transition Conditions section begins with the CONDITIONS keyword and specifies the combination of input signals that determine the transition conditions.

For example, BUSREQ is a transition condition that is true when (1) M_IO is high and INTCYC and A20 are low or when (2) M_IO and A20 are low and INTCYC is high.

Transition conditions entries are combinatorial Boolean equations. Register/latch equations cannot be used to specify transition conditions.

Mealy State Machine Example

Figure 4-26 is a state diagram for a Mealy Machine status checker. This machine transitions between three states. The outputs of the machine, however, may be set to different values for a given state, depending on input conditions. A Moore Machine implementation of this design would require nine states to accomplish the same task (one state for each possible combination of outputs). Figure 4-27 is a PLDasm listing of the design.

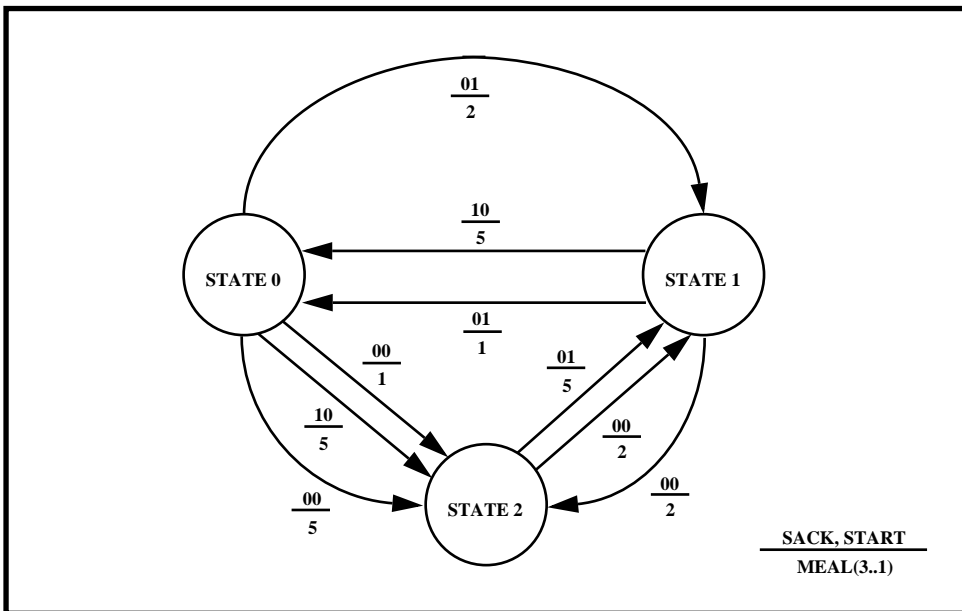


Figure 4-26 State Diagram for Mealy Status Checker

```

Title      Shows Mealy Machine
Pattern    pds
Revision
Author     Altera Applications
Company    Altera
Date       October 1994

CHIP exmealy1 EP224
;
; Design is a system state tracker.
;

PIN CLOCK ; clock for state variables
PIN SACK ; input conditions
PIN START

PIN MEAL3 ; Mealy-machine outputs
PIN MEAL2
PIN MEAL1
PIN MSTATE1 ; Mealy state variables
PIN MSTATE0

STRING M1_OUT '/MEAL3 * /MEAL2 * MEAL1'
STRING M2_OUT '/MEAL3 * MEAL2 * /MEAL1'
STRING M5_OUT ' MEAL3 * /MEAL2 * MEAL1'

STATE MEALY_MACHINE

DEFAULT_OUTPUT /MEAL3 /MEAL2 /MEAL1

DEFAULT_BRANCH HOLD_STATE
; state assignments

STATE0 = /MSTATE1 * /MSTATE0 ; powerup state
STATE1 = /MSTATE1 * MSTATE0
STATE2 = MSTATE1 * /MSTATE0

; state transitions, which state to go to next

STATE0 := TRIG0-> STATE2
+ TRIG1  -> STATE1
+ TRIG2  -> STATE2

STATE1 := TRIG0-> STATE2
+ TRIG1  -> STATE0
+ TRIG2  -> STATE0

STATE2 := TRIG0-> STATE0
+ TRIG1  -> STATE1
+ TRIG2  -> STATE1

```

Figure 4-27 Mealy State Machine Example

```

; output transitions, what the output variables should be

STATE0.OUTF = TRIG0-> M1_OUT
+ TRIG1  -> M2_OUT
+ TRIG2  -> M5_OUT

STATE1.OUTF = TRIG0-> M2_OUT
+ TRIG1  -> M1_OUT
+ TRIG2  -> M2_OUT

STATE2.OUTF = TRIG0-> M5_OUT
+ TRIG1  -> M5_OUT
+ TRIG2  -> M1_OUT

; conditions that determine transitions

CONDITIONS
TRIG0 = /SACK * /START
TRIG1 = /SACK * START
TRIG2 = SACK * /START

EQUATIONS

MSTATE0.CLKF = CLOCK; hook clock up, for device; independence
MSTATE1.CLKF = CLOCK

```

Figure 4-27 Mealy State Machine Example (Continued)

Simulation

This section shows how to write a simulation section to functionally simulate your design. Functional simulation of designs is optional, but is recommended to make sure that your design works the way you intend it. PLDasm provides the following simulation capabilities:

- Event-driven simulation of combinatorial, registered, and state machine designs
- Ability to set any input, preload any register, and compare any output against an expected value
- Ability to group signals together (form a vector) to simulate a bus
- FOR-TO and WHILE loops
- IF-THEN-ELSE control
- Generation of test vectors from simulation results for inclusion in the JEDEC file
- Simulation history file with ability to output a subset of signals to a secondary trace file

Simulation is specified in the Simulation section of PLDasm files. The start of the Simulation section is indicated by the keyword SIMULATION. PLDasm automatically executes the simulation section when it is present and simulation is enabled from the PLDshell Plus Compile/Sim submenu.

Figure 4-28 is a sample Simulation section of a PLDasm file. It shows most of the simulation syntax and is used as the example in the following discussion. Syntax is discussed under two headings: (1) basic commands and (2) flow control.

Basic Commands

SETF — Sets the designated inputs, states, or vectors to the specified value. This is the command that allows you to change input or feedback values. Simple simulation of combinatorial designs can be accomplished using SETF commands alone.

CLOCKF — Clocks registers high-then-low, or low-then-high. A clock initialized to a low (e.g., SETF /CLK), goes high-then-low. A clock initialized to a high (e.g., SETF CLK), goes low-then-high. Simple simulation of registered circuits can be accomplished using CLOCKF and SETF commands alone.

PRLDF — Preloads the designated registers with the specified high or low values. This is a quick way to set state machines and other registered designs to known states. Clock and control signals (OE, Clear, Preset, etc.) should be set to a known state before executing the preload command; if this is not followed, preloaded registers will immediately change to an unknown state. Figure 4-28 shows how to use this command. See “Test Vector Notes” for guidelines on using this command.

VECTOR — Assigns a group of signals to a variable name. Makes it easy to group bus signals or like signals together. This is a *superset* feature of PLDasm syntax. Precedence of output is [msb . . . lsb]. Since vector assignments are declarations, they must come first in the simulation section. Figure 4-28 shows how to use this command.

TRACE_ON and **TRACE_OFF** — By default, all inputs and outputs to a device are included in the simulation history file (.HST). If you wish to observe a subset of these signals in a secondary trace file (.TRF), use the TRACE_ON/TRACE_OFF sequence. This command sequence opens and closes a simulation trace file (.TRF) to store the specified signals. These commands can be placed anywhere in the Simulation section, but only one pair is allowed. TRACE_ON can be used with a signal and/or vector list to specify which are to be recorded in the trace file. In Figure 4-28, inputs IN1 and IN2, CLK, and outputs Q0 through Q3 are to be recorded in the trace file. The OE input is not included.

```

SIMULATION

; send some signals to trace file, define vector
; preload registers and set inputs to known state

VECTOR NUM := [ Q3, Q2, Q1, Q0 ]
TRACE_ON CLK IN1 IN2 Q0 Q1 Q2 Q3
SETF OE /CLK /IN1 /IN2
PRLDF /Q3 /Q2 /Q1 /Q0; clock and controls set
                                ; before preload command

; count 9 times, disable OE between 3 and 5

FOR j := 1 TO 9 DO
  BEGIN
    IF ( j = 3 ) THEN
      BEGIN
        SETF /OE
      END
    IF ( j = 5 ) THEN
      BEGIN
        SETF OE
      END
    CLOCKF CLK
  END

; check other combinations of inputs

SETF IN1 IN2
CLOCKF CLK
SETF /IN1 IN2
CLOCKF CLK
SETF IN1 /IN2
CLOCKF CLK

; close trace file

TRACE_OFF

; end simulation

```

Figure 4-28 Sample Simulation Section

CHECK — Checks the designated signals for the specified logic state and displays an error if the actual state and expected state are not the same. Figure 4-29, Additional Simulation Syntax Examples, shows how to use this command.

Flow Control

BEGIN/END — Delimits a sequence of simulation commands; used within a conditional statement. Figure 4-29 shows examples of this syntax.

```

; FOR/TO/THEN loop including CHECK command

; loop 8 times, check for high on Q2 after 4th clock

    FOR k := 1 TO 8 DO
        BEGIN
            CLOCKF CLK
            IF ( k = 4 ) THEN
                BEGIN
                    CHECK Q2
                END
            END
        END

; WHILE loop example

; loop until OUTA and OUTB are both low

    WHILE ( OUTA + OUTB ) DO
        BEGIN
            CLOCKF ACLK1
        END

; IF/THEN/ELSE example

; loop 16 times, check for READY = low on 5th and 6th
; clocks; check for READY = high all other times

    FOR ((cntr >= 4) * (cntr <= 5))
        BEGIN
            IF cntr (>=4 * <=5) THEN
                BEGIN
                    CHECK /READY
                END
            ELSE
                BEGIN
                    CHECK READY
                END
            END
            CLOCKF CLK
        END
END

```

Figure 4-29 Additional Simulation Syntax Examples

FOR/TO/DO <loop> — FOR statement defines conditions under which the DO loop is executed. Figure 4-29 shows an example of this syntax.

WHILE/DO <loop> — WHILE statement defines conditions under which the DO loop is executed. Figure 4-29 shows an example of this syntax.

IF/THEN/ELSE — Provides IF/THEN/ELSE syntax for executing sequences of simulation commands. Conditions must be enclosed in parentheses. Figure 4-29 shows an example of this syntax.

ASSIGNMENTS — Vectors may be assigned any numeric constant or any Boolean expression in conjunction with a SETF command. Constants may be expressed in any

of the following number systems: hexadecimal, octal, binary, and decimal. Examples of how to specify different number systems are as follows:

| | | | |
|-------------|---------------------|----|--------|
| hexadecimal | leading | 0x | 0xD51B |
| | or | #h | #hD51B |
| | or | #H | #HD51B |
| octal | leading | 0 | 0377 |
| | or | #o | #o377 |
| | or | #O | #O377 |
| binary | leading | #b | #b1011 |
| | or | #B | #B1011 |
| decimal | no leading sequence | + | 76 |

For example:

```
SETF VECTOR := 0xff      ; hexadecimal assignment
SETF VECTOR := a * b    ; decimal assignment
```

An example of how to use this feature is shown below:

```
;set all address of decode
VECTOR ADDR := [ a8, a7, a6, a5, a4, a3, a2, a1 ]

FOR COUNT := 0 TO 256 DO
  BEGIN
    SETF ADDR := COUNT
  END
```

EXPRESSIONS — WHILE/DO and IF/THEN/ELSE allow any combination of Boolean expression or conditional for the condition. For example:

```
IF ((A * B) +=: (C + D) THEN
```

CONDITIONALS — The following conditional operations are supported:

| | |
|----|--------------------------|
| = | Equals |
| > | Greater than |
| < | Less than |
| >= | Greater than or equal to |
| <= | Less than or equal to |
| /= | Does not equal |

The orders of precedence for simulation of the Boolean and conditional operators are as follows:

Figure 4-1 Boolean

| | | |
|---------|-----|----------|
| Highest | / | Negation |
| | * | AND |
| | :+: | XOR |
| Lowest | + | OR |

Table 4-1 Conditional

| | | |
|---------|--------------|------------|
| Highest | >, >=, <, <= | Relational |
| Lowest | =, <>, /= | Equality |

The order is left to right for operators on the same level of precedence. Parentheses may be used to force a particular order.

Access to All Internal Signals

In order to access all output/control signals for macrocells and SRAM blocks (to check/view their values during simulation), the following signal name extension forms are accepted in the simulation section:

| | |
|----------|----------|
| OUT.SETF | OUT.RSTF |
| OUT.TRST | OUT.CLKF |
| OUT.ACLK | OUT.LE |
| OUT.ALE | OUT.WE |
| OUT.BE | OUT.ADDR |
| OUT.DATA | OUT.CMP |
| OUT.D | OUT.T |
| OUT.J | OUT.K |
| OUT.S | OUT.R |

Chapter 5 — Compiling, Simulating, and Estimating

This section describes design methodology for both device-independent and device-dependent designs and shows how to use the compilation, estimation, and simulation features of PLDshell Plus/PLDasm.

Design Methodology

There are two fundamental approaches to PLD design: device-independent design and device-dependent design.

With device-independent design, a state diagram or architectural specification is first translated into a PLDasm file without specifying a target device. The design can be simulated and reworked until it is functionally correct. Once the design is working, the full compilation process can be executed to fit the design into a target device and generate a JEDEC file. Simulation can be run during this stage to test any changes made to help fit the design into a device.

The major benefit of device-independent design is that it allows the designer to concentrate on the conceptual aspect of the design. Some of the choices affecting the device selection include available packages, cost and device performance. This approach is a compromise because some designs can function correctly, but may not be able to fit into a specific device architecture without logic changes.

With device-dependent design, the target device is selected at the start. Creation of the design and fitting of the design occur together. Since a designer will already be familiar with the architecture that has been chosen, any unique features of that architecture can be utilized as the design is developed. Fitting occurs during each pass through the compiler.

The major benefit of device-dependent design is that the designer knows that if the design compiles, it has already been fitted. The trade-off for this approach is that the full fitting process must be run on each pass through the compiler.

Each approach is discussed in greater detail in the following sections.

Device-Independent Design

To take advantage of device-independent design, the design is specified without identifying a target device (the ALTERA_ARCH keyword is used for the device name in the CHIP declaration). During processing of the design by the compiler, no device-specific checks, optimizations, or conversions will be performed, and the fitter is not executed. All other supported compiler operations are available, such as minimization, D/T synthesis of state machines, simulation, etc. The processing flow for both device-independent design and device-dependent design is shown in Figure 5-1.

NOTE:

INTEL_ARCH is accepted as an alternative for ALTERA_ARCH, for backward compatibility with existing designs.

Once the design is syntactically correct and the simulation shows the implementation is satisfactory, the estimation report (<design_name>.EST) produced by device-independent compilation may be consulted for guidance in selecting a target device. The design may be targeted to the desired device by specifying the device name in the CHIP declaration. This will automatically cause the design to be processed in a device-dependent manner, with the compiler applying device-specific checks, optimizations, and/or conversions.

Note that it is quite possible to implement a design in a device-independent manner which may not fit into the device chosen for device-dependent compilation. Some adjustment of the design may therefore be needed to obtain a successful fit in the target device. While doing device-independent design, it is good practice to keep in mind the architecture of the device(s) of interest for the final implementation to minimize the number of adjustments once the final device is chosen.

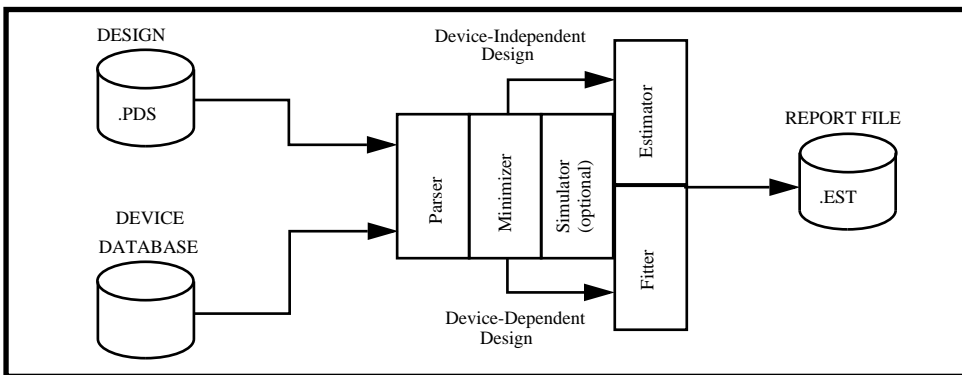


Figure 5-1 Estimator Processing Flow

Compiling, Simulating, and Estimating

Sample Design

Often a design is begun with a state diagram of the design. An example of this is the state diagram shown in Figure 5-2. This is a 4-bit up/down counter featuring Reset and Set inputs. The design is implemented as a Moore State Machine.

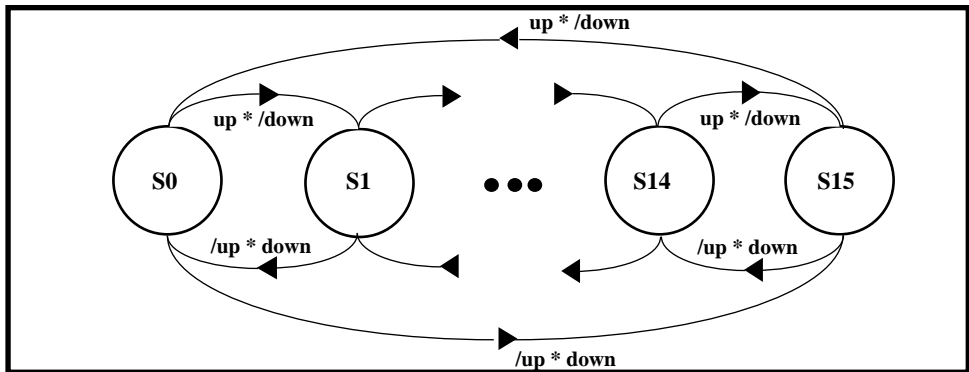


Figure 5-2 4-Bit Up-Down Counter State Diagram

Figure 5-3 is an example PLDasm listing of the design. Note that in the Declaration section the CHIP declaration includes the reserved device name ALTERA_ARCH.

```

Title          4 Bit Up/Down and Fast/Slow Counter
Pattern        none
Revision       1.0
Author         John Doe
Company        Altera
Date           10-19-94

CHIP updown    ALTERA_ARCH

PIN  clock      ; clock input
PIN  up         ; count up
PIN  down       ; count down
PIN  fast       ; /fast -> count regular :: fast -> count by two
PIN  reset      ; reset flip-flops
PIN  set        ; set flip-flops
PIN  q0         ; lsb
PIN  q1         ; .
PIN  q2         ; .
PIN  q3         ; msb

STATE MOORE_MACHINE
s0 = /q3 * /q2 * /q1 * /q0      s1 = /q3 * /q2 * /q1 * q0
s2 = /q3 * /q2 * q1 * /q0      s3 = /q3 * /q2 * q1 * q0
s4 = /q3 * q2 * /q1 * /q0      s5 = /q3 * q2 * /q1 * q0
s6 = /q3 * q2 * q1 * /q0      s7 = /q3 * q2 * q1 * q0
s8 = q3 * /q2 * /q1 * /q0      s9 = q3 * /q2 * /q1 * q0
sa = q3 * /q2 * q1 * /q0      sb = q3 * /q2 * q1 * q0
sc = q3 * q2 * /q1 * /q0      sd = q3 * q2 * /q1 * q0
se = q3 * q2 * q1 * /q0      sf = q3 * q2 * q1 * q0
se = q3 * q2 * q1 * /q0      sf = q3 * q2 * q1 * q0

```

Figure 5-3 Sample Device-Independent Design

```

s0:= upby2 -> s2 + dnby2 -> se + upby1 -> s1 + dnby1 -> sf
s1:= upby2 -> s3 + dnby2 -> sf + upby1 -> s2 + dnby1 -> s0
s2:= upby2 -> s4 + dnby2 -> s0 + upby1 -> s3 + dnby1 -> s1
s3:= upby2 -> s5 + dnby2 -> s1 + upby1 -> s4 + dnby1 -> s2
s4:= upby2 -> s6 + dnby2 -> s2 + upby1 -> s5 + dnby1 -> s3
s5:= upby2 -> s7 + dnby2 -> s3 + upby1 -> s6 + dnby1 -> s4
s6:= upby2 -> s8 + dnby2 -> s4 + upby1 -> s7 + dnby1 -> s5
s7:= upby2 -> s9 + dnby2 -> s5 + upby1 -> s8 + dnby1 -> s6
s8:= upby2 -> sa + dnby2 -> s6 + upby1 -> s9 + dnby1 -> s7
s9:= upby2 -> sb + dnby2 -> s7 + upby1 -> sa + dnby1 -> s8
sa:= upby2 -> sc + dnby2 -> s8 + upby1 -> sb + dnby1 -> s9
sb:= upby2 -> sd + dnby2 -> s9 + upby1 -> sc + dnby1 -> sa
sc:= upby2 -> se + dnby2 -> sa + upby1 -> sd + dnby1 -> sb
sd:= upby2 -> sf + dnby2 -> sb + upby1 -> se + dnby1 -> sc
se:= upby2 -> s0 + dnby2 -> sc + upby1 -> sf + dnby1 -> sd
sf:= upby2 -> s1 + dnby2 -> sd + upby1 -> s0 + dnby1 -> se

```

CONDITIONS

```

upby1 = /fast * up * /down dnby1 = /fast * /up * down
upby2 = fast * up * /down dnby2 = fast * /up * down

```

EQUATIONS

```

q0.clkf = clock q1.clkf = clock q2.clkf = clock q3.clkf = clock
q0.rstf = reset q1.rstf = reset q2.rstf = reset q3.rstf = reset
q0.setf = set q1.setf = set q2.setf = set q3.setf = set
q0.trst = vcc q1.trst = vcc q2.trst = vcc q3.trst = vcc

```

SIMULATION

```

VECTOR count := [q3,q2,q1,q0]

```

```

SETF /fast up /down /reset /clock
PRLDF /q0 /q1 /q2 /q3
SETF clock
SETF reset /clock
SETF /reset set
SETF /reset /set
CLOCKF clock

```

```

;count up to 15 by 1 and roll over

```

```

FOR x:=1 to 17 DO
BEGIN
CLOCKF clock
END

```

```

;test nop (up and down=vcc)

```

```

SETF /fast up down
FOR x:=1 to 9 DO
BEGIN
CLOCKF clock
END

```

Figure 5-3 Sample Device-Independent Design: UPDOWN.PDS (Continued)

```

;test reset and set
SETF /down
CLOCKF clock
CLOCKF clock
CLOCKF clock
CLOCKF clock
SETF reset
FOR x:=1 to 4 DO
BEGIN
    CLOCKF clock
END
SETF /reset set
FOR x:=1 to 4 DO
BEGIN
    CLOCKF clock
END

```

Figure 5-3 Sample Device Independent Design: UPDOWN.PDS(Continued)

Now that the design is created, to simulate it, just compile it using the Simulate Only processing option in the Compile/Sim menu. Once the design has been implemented in a PLDasm source file, it can be processed through the parser, minimizer (optional), estimator (optional), and simulator.

For the purposes of this discussion, only the first part of the simulation section is shown. (The file called UPDOWN.PDS, which appears in Figure 5-3, also appears in the PLDshell examples directory. It contains the complete simulation section.)

Figure 5-4 shows the waveforms produced by this simulation. Note that the outputs (Q0-Q3) go high immediately after SET goes high, thus implementing an asynchronous Set. Reset is also asynchronous, clearing the outputs to a logical low immediately after being asserted.

During device-independent design, all Set and Reset signals are asynchronous. Depending upon which device is eventually selected, the Set signal may be implemented synchronously. Differences between device-independent and device-dependent design are described in “Device-Independent Design Notes.”

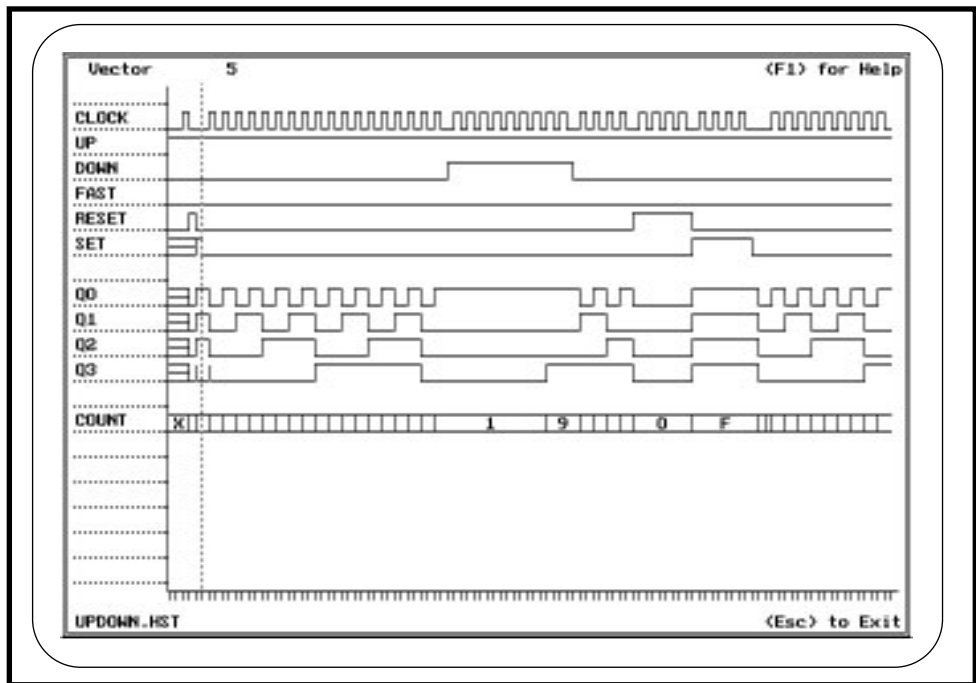


Figure 5-4 Device-Independent Design Waveforms (UPDOWN.HST)

Device-Independent Design Notes

The following notes are provided to help you perform device-independent design:

1. Device-independent design allows you to design with no device resource limitations. This means that you can implement options that are not supported on all devices. Some redesign for fitting may eventually be needed when a device is finally selected. Using the estimator will help with the final device selection. Areas to be aware of include the following:
 - **Set and Reset Signals** — These are asynchronous during device-independent processing; only those actually supported by the device are valid during device-dependent processing.
 - **Register Types** — Any register type (D, T, JK, SR) is valid during device-independent processing; only those actually supported by the device are valid during device-dependent processing.
 - **Input Types** — Any input type (direct, latched, registered) is valid during device-independent processing; only those actually supported by the device are valid during device-dependent processing.
 - **Register Preloads** — Register preloads are affected by the following three variables:

- Whether the device selected for your design supports preloads (EP22V10 and EP22V10E support preloads)
- Whether you see the effect of a preload in simulation
- Whether preload test vectors are output to a JEDEC file

For device-independent designs, the question of device support and viewing simulation are not an issue. But you can note that preload simulation vectors are output to the JEDEC file.

For device-dependent designs, if the target device supports preloads, you will see the preload during simulation and the simulation test vectors are output to the JEDEC.

However, for device-dependent designs with a target device that *doesn't* support pre-loads, you still see the preload during simulation, and the preloads are handled as follows:

- The first preload is converted to SETF
 - All subsequent preloads are not included in the test vectors
- Equation Size — Any size equation can be implemented during device-independent processing (limited only by the system memory); during device-dependent processing, equation size is limited to the number of p-terms in the target device.
2. Large state machines can translate into designs that contain large numbers of p-terms for equations. If the minimizer is not run when Simulate Only is selected (in the Compile/Sim Menu), these designs may take a long time to simulate. In these cases, it may be better to enable the minimizer via the Compile Options submenu. Simulations will run much faster when equations are minimized.

Device-Dependent Design

For the device-dependent design, the same design as shown in Figure 5-3 will be used with a few changes. The reserved device name ALTERA_ARCH is replaced with EP22V10. Figure 5-5 shows the waveforms resulting from the device-dependent simulation. Note that the outputs (Q0-Q3) don't go high until the rising edge of the second clock (synchronous preset). The synchronous input behaves exactly as the EP22V10 operates; this stands in contrast to the asynchronous input for the device-independent version of the design.

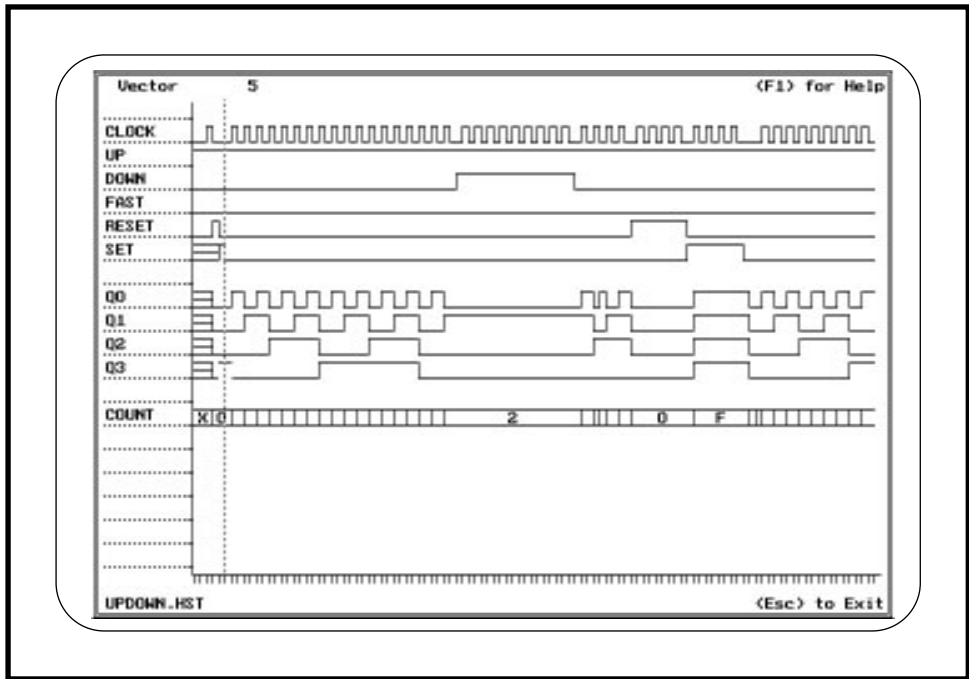


Figure 5-5 Device-Dependent Design Waveforms (UPDOWN.HST)

Device-Dependent Design Notes

When performing device-dependent design, the compiler and simulator operate with full knowledge of the target device characteristics. The following notes apply to compilation when the target device is selected.

For All Designs

- Security Bit(s)** — Most PLDs have a Security Bit (sometimes called Verify Protect). The default state of the Security Bits during compilation is off, which allows programmed devices to be read. When this bit is set ON, the contents of the device cannot be read. This provides a high degree of design security. To set this bit in the JEDEC file, you must specify SECURITY=ON in the PLDasm source file. (The erased state of Altera PLDs is SECURITY=OFF.)
- Turbo Bit(s)** — Most Altera PLDs contain one or more Turbo Bits that optimize device performance for speed or power savings. The default state of the Turbo Bits during compilation is On, which optimizes device performance for speed. If you wish to optimize a device for power savings, you must specify TURBO=OFF in the PLDasm source file. (The erased state of Altera PLDs is TURBO=OFF.)
- Device Conversion** — Some logic compilers automatically treat outputs and equations as active low for devices with fixed, active-low outputs (e.g., a 16L8), even

when the equation or signal name does not include the slash (/) prefix. Because Altera PLDs contain programmable outputs, the presence of the slash prefix is always required to designate an equation or output as active low according to PLDasm syntax. The absence of the prefix always means active high.

For PAL/GAL Designs

- **Non-Standard PAL/GAL Pin-outs** — Different PLCC pinouts for devices have been adopted by some PAL/GAL manufacturers. PLDasm is only able to fit designs that use the standard pinout supported by Altera’s PLCC packages. You may need to change the pin numbers in your source file when compiling designs that use PAL/GAL PLCC device names.
- **20-/28-Pin Package Designations** — PLDasm does not recognize all variations of PLCC package designators for all PAL/GAL manufacturers. To ensure recognition of PLCC names in PLDasm source files, use an “NL” designator for 20-pin packages and an “FN” designator for 28-pin packages after the base part name.
- **Automatic OE Inversion** — In some cases, PLDasm performs modifications to make sure that the final design performs in the same way as the original design. For example, registered PALs/GALs contain a dedicated active-low OE pin. Altera PLDs, however, use a p-term to control the OE, even for registered designs. In this case, the software generates the appropriate p-term for an active-low OE signal. If OE is always enabled (tied to GND) in registered PAL/GAL designs, it will be always enabled in the Altera PLD design (tied to VCC). OE inversion is performed automatically; you do not need to make this change yourself. When modifications like this occur, a message is displayed on the screen and a comment is placed in the PLDasm source file.

Using the PLDasm Compiler Options

The Compile Options Menu offers a set of compiler options that can handle a variety of design considerations. Figure 5-6 shows the Compile Options submenu and the default settings for each option.

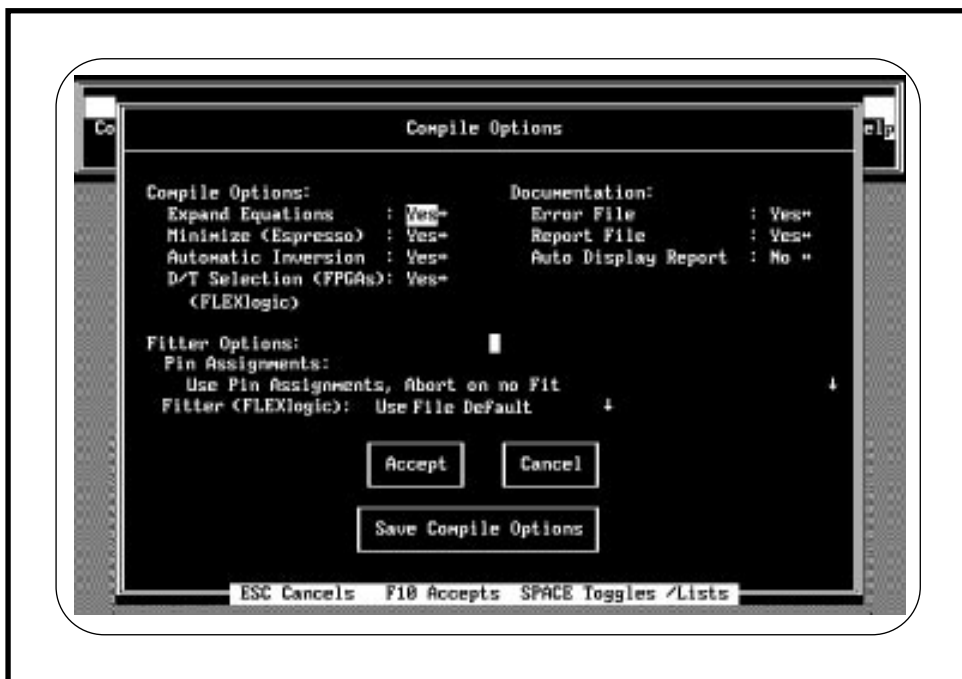


Figure 5-6 Compile Options Submenu

Table 5-1, Valid Compiler Option Combinations, lists all the valid combinations of compiler options and what each combination produces.

NOTE:

When compiling from the DOS command line, these options are available, but in different form. See Appendix C to select options from the command line.

Table 5-1 Valid Compiler Option Combinations

| Expand Equations | Minimize (Espresso) | Automatic Inversion | Description |
|---|---------------------|---------------------|---|
| No | No | No | Do not change equations. Use this option if you have written your equations with hazard coverage in mind. |
| Yes | No | No | Expand to SOP Only |
| Yes | Yes | No | Expand, Minimize, No Automatic Inversion |
| Yes | Yes | Yes | Expand, Minimize, Automatic Inversion (Default options) |
| Yes | No | Yes | Expand, no Minimization, Automatic Inversion |
| All other combinations are <i>invalid</i> | | | |

Compile Options

Expand Equations:

Default — Uses the EXPAND option value as defined in the .PDS file. If the option is omitted in the file, the value of EXPAND=ON will be used.

Yes — Forces the .PDS file option EXPAND=ON, overriding any value in the .PDS file. All designs must be in SOP form for minimization and fitting. In cases where equations are already in SOP form and you do not want the compiler to make any changes, set this option to NO.

No — Forces the PDS option EXPAND=OFF, overriding any value in the .PDS file. The option of EXPAND=OFF is provided for designs with equations already in SOP form which the designer does not want altered in any way. All designs must be in SOP form for minimization and fitting. Since the minimization and automatic inversion algorithms require SOP form, the combination of Expand Equations = NO with Minimize = YES and/or Automatic Inversion = YES is illegal.

Expand example:

Original: $A = x * (y + z)$

Expand: $A = x * y$
 $+ x * z$

Minimize: $A = x * y$
 $+ x * z$

Minimize (Espresso):

Default — Uses MINIMIZE option as defined in the .PDS file. If the option is omitted in the file, the value of MINIMIZE=ON will be used.

Yes — Forces MINIMIZE=ON, overriding any value in the .PDS file. PLDasm will minimize SOP equations to the least number of p-terms. The minimizer uses the ESPRESSO-IImv logic reduction algorithm.

No — Forces MINIMIZE=OFF, overriding any value in the .PDS file. Use only when a design is known to fit and no changes should be made by the compiler.

Minimization example:

Original: $A = x * y * z$
 $+ x * /y * z$
 $+ /x * y * z$

Expand: same

Minimize: $A = x * z$
 $+ y * z$

Automatic Inversion:

Default — Uses INVERSION option as defined in the .PDS file. If the option is omitted in the file, the value of INVERSION=ON will be used. This option controls whether PLDasm can invert SOP equations using DeMorgan's Law.

Yes — Forces INVERSION=ON, overriding any value in the .PDS file. Automatically inverts equations using DeMorgan's equations if the inverted form will use fewer p-terms.

No — Forces INVERSION=OFF, overriding any value in the .PDS file. When automatic inversion is not selected and the minimizer is run, DeMorgan's inversion rules can still be applied, but only at your discretion. The minimizer determines if the inverted form requires less p-terms than the original form. If so, the minimizer requests permission to use the inverted form. You can respond "Y" or "N".

NOTE:

If you set Automatic Inversion to No (OFF), the expansion step can take *significantly* longer.

D/T Selection (FLEXlogic):

This option is used with FLEXlogic devices to generate "T"-type flip-flop inputs whenever a "D"-type flip-flop is specified, and vice-versa. The device fitter then selects whichever form of the flip-flop uses the smallest number of product terms.

Default — Uses DT_SYNTHESIS option as defined in the .PDS file. If the option is

omitted in the file, the value of DT_SYNTHESIS=ON will be used for FLEXlogic devices, and DT_SYNTHESIS=OFF for all other devices.

Yes — Forces DT_SYNTHESIS=ON, overriding any value in the .PDS file. This option is used with FLEXlogic devices to generate “T”-type flip-flop inputs whenever a “D”-type flip-flop is specified, and vice-versa. One disadvantage is that this can generate very large equation files which require much CPU time to process, thus the compile process runs slower.

No — Forces DT_SYNTHESIS=OFF, overriding any value in the .PDS file. Use NO if you do not need the reduced D/T equations or you do not want PLDasm to change the Flip-flops you have specified. Or, you can use this if PLDasm runs out of memory on your computer.

Documentation

Error File:

- **Yes (default)** — Creates an error file (.ERR) each time the compiler or simulator is run. The error file contains all the messages displayed during the compilation/simulation process, including information and warning messages. This file can be viewed under the View Menu. The on-line help for each error message can be accessed by moving the cursor onto the specific error message and pressing <F10>.
- **No** — No error file generated. Use if disk space is limited.

Report File:

- **Yes (default)** — Creates the utilization report file (.RPT) each time the compiler is run. The report file shows the final pinout of the design and shows how resources are assigned in the final, fitted design. It also shows which resources could not be fitted and why.
- **No** — No report file is generated.

NOTE: The report file is different for each family, but will typically contains the following sections:

- **Error Report** - A listing of error messages generated by the PLDasm compiler.
- **Part Description** - Contains device pinout diagram and signal name-to-pin mapping.
- **Pin Assignments** - Lists outputs, inputs, and unused device resources. A description of each field in the pin assignments listing follows:
 - **Signal** - Signal name.
 - **Pin** - Pin number.

- **V** - Volts. Default is 5V; 3.3V indicated by 3.
- **D** - Drain. If a signal's output type is open drain, an O will appear in this column for that signal.
- **Type** - Register type. Possible register types are C = Combinatorial; D = D-Type Register; T = T-Type Register; C(RAM); or D(RAM).
- **PTerms** - Number of p-terms used.
- **[nu, up, lo, nl]** - Lists adjacent pterms. nu = next upper pterm; up = upper pterm; lo = lower pterm; nl = next lower pterm.
- **Blk** - CFB (Configurable Function Block) number.
- **MCell** - Macrocell number.
- **CFB Information** - Lists in which CFB an output and input is placed. Also lists fan-ins per CFB.
- **RAM Descriptions** - Lists which signals are being used for SRAM (and the data lines going into each).
- **Design Statistics** - Statistics listed on I/Os, Macrocells, P-terms, and SRAM blocks.

Auto Display Report:

- **Yes** — After the selected processes have finished, the most recent report generated is displayed on the screen. This saves keystrokes since you do not need to go to the View Menu and select the file. Particularly when estimating a design's requirements, this option can save time.
- **No (default)** — The auto report display function is disabled.

Fitter Options

The Altera PLDasm fitter uses a graduated pin-reassignment approach for the FLEXlogic devices it supports. There are six fitter options located in the Pin Assignments field of the Compile Options submenu. They are:

- **Use Defaults, as Specified in Design File (default)**

This option will force the fitter to use the defaults specified in the .PDS design file only and aborts with an error message if a fit is not possible.

- **Use Design and Previous Pin Assignments, Abort on no Fit**

This option is used when the final pinout for a design is already fixed (e.g., the target board is already laid out). The fitter will try to fit all resources to the pins declared in the source file and from any previous compile efforts. If any pin does not fit, the process aborts with an error message. This is the default fitting option.

- **Use Design and Previous Pin Assignments, but Reassign if Needed**

This option will allow the fitter to try and fit the design from the pin assignments specified in the .PDS file and from previous compile efforts. If a fit is not possible, the fitter will utilize an auto-fit algorithm and ignore the user specified pin assignments. The device fitters use a graduated pin-reassignment approach before going to a complete device auto-reassign. This allows as many original pin assignments to remain unchanged as possible.

- **Use Design Pin Assignments, but not Previous, Abort on no Fit**

This option will allow the fitter to use the pin assignments specified in the design file but not from previous compile efforts. It will abort with an error message if a fit is not possible.

- **Use Design Pin Assignments, but not Previous, and Reassign if Needed**

This option will allow the fitter to try and fit the design from the pin assignments specified in the .PDS file but not from previous compile efforts. If a fit is not possible, the fitter will utilize an auto-fit algorithm and ignore the user specified pin assignments.

- **Ignore all Pin Assignments**

This option requires the fitter to use its automatic fitting algorithms to fit a design. It can eliminate the need for the designer to know all the details of the device pinout. All pin assignments in the source file are ignored. The fitter will abort if all attempts have failed. Note that the automatic fitting algorithms make some assumptions about the information provided for each output. These assumptions can be changed (see “Implementing Alternate I/O Options” in Chapter 4).

NOTES:

If the pin is not on the left side of an equation, the fitter makes the pin into an input.

Using any choice other than the ‘Use PDS Default’ will override the values of any FITTER_PINS or FITTER_PREV_PINS options that may be set in the PDS file.

Fitter (FLEXlogic):

Use File Default — Uses the default value of the FITTER_ALGORITHM option in .PDS file, if any is specified.

Normal Fit — Forces FITTER_ALGORITHM=NORMAL, overriding any value in the .PDS file.

Complete Fit — Forces FITTER_ALGORITHM=COMPLETE, overriding any value in the .PDS file. Use only when a design has problems fitting, as this can take a long time to run. See Chapter 4 for more information on options.

Using the Simulation Options

The Simulation options allow the designer to show asynchronous events during stabilization of outputs during each simulation step and set the maximum number of such events. This is very useful in identifying race/glitch conditions in combinatorial or asynchronously clocked designs and in tracking design problems in circuits that make extensive use of feedbacks.

Show Asynch. Events

Normally set to OFF (No). This option is normally not used unless a design problem is suspected.

Turn on when you want to locate and analyze problems in the design (for example, to analyze why outputs do not appear to function correctly). Can also be used to observe the detailed operation of asynchronously clocked state machines.

Max. Asynch. Events

Normally set to 32 events. The threshold range is 1 to 32,767 simulation events. The default value is normally adequate to analyze a design problem. Adjust the threshold value as necessary to display the number of events to provide enough detail to analyze the problem. For example, a design with numerous asynchronous events should use a larger number.

Viewing Simulation Output Files

The PLDasm simulator generates two output files: (1) a history file (.HST) containing simulation results for all signals for the entire simulation, and (2) an optional trace file (.TRF) containing simulation results for the specified signals for a defined part of the simulation.

The .HST file can be viewed as state table (1s and 0s) or as waveform output under the View Menu. Waveforms are viewed in graphics mode and printed in text mode. Text mode printing uses standard ASCII characters or the extended IBM/DOS graphic character set. Figure 5-7, State Table Output (.HST file), shows simulation output in a state table format.

The .TRF file can be used to isolate the critical signals you need to check during simulation. Simulation generates test vectors which can be put in the JEDEC file which will stimulate the device at programming time so you can functionally test the device.

```

; position 1CLKS _____ SIGNAL NAMES AND
; position 2IN1S _____ POSITIONS FROM
; position 3IN2S _____ HISTORY FILE
; position 4OE S
; position 5Q0 S
; position 6Q1 S
; position 7Q2 S
; position 8Q3 S
; position 9NUMM 1
;
; CII0 QQQQ N
; LNNE 0123 U
; K12 M _____ SIGNAL NAMES
;
; _____
0001 XXXX X
0001 LLLL 0
1001 HLLL 3
0001 HLLL 3
1001 LHHL 6
0001 LHHL 6
1001 HLHL 5
0001 HLHL 5
0000 ZZZZ Z
1000 ZZZZ Z
0000 ZZZZ Z
1000 ZZZZ Z
0000 ZZZZ Z
0001 HLLL 7 — INPUTS
0001 HLLL 7 — OUTPUTS
1001 LHLH A
0001 LHLH A
1001 HLLH 9 — VECTORS
0001 HLLH 9
1001 LLLH 8
0001 LLLH 8
1001 HHLH B
0001 HHLH B
0111 HHLH B
1111 LLLL 2
0111 LLLL 2
0011 LLLL 2
1011 HLLL 3
0011 HLLL 3
0101 HLLL 3
1101 LLLL 2
0101 LLLL 2

```

Figure 5-7 State Table Output (.HST file)

Simulation Notes

The following notes describe some behaviors of the simulator that may be important for certain types of designs:

Synchronous vs Asynchronous Clocking Behavior

1. If you are using synchronous or asynchronous registers you will see several different behaviors if the data changes on a clock edge:
 - For synchronously clocked registers (i.e., registers clocked by the dedicated clock pin), the data presented to the register input will not be clocked through the register during the current clock cycle. This behavior emulates the synchronous clocking behavior of PLDs, which require that data meet a setup time to the clock edge.
 - For asynchronously clocked registers (i.e., registers clocked by a p-term from the logic array), the data presented to the register input will be clocked through the register during the current clock cycle. The behavior emulates the asynchronous clocking behavior of PLDs, which have a much shorter (or no) setup time to the clock edge.
2. Executing the SETF command on a vector without an assignment sets all bits of the vector to the specified logic level (logical 1 or 0). For example, the following sequence declares a vector (NUM), sets all bits of NUM to 1, then clears all bits of NUM to 0.

```
VECTOR NUM = [ Q3 Q2 Q1 Q0 ]  
SETF NUM  
SETF /NUM
```

A more appropriate way is to assign the vector as follows:

```
VECTOR NUM = [ Q3 Q2 Q1 Q0 ]  
SETF NUM := 0xF  
SETF NUM := 0x0
```

3. A .FB extension can be used to simulate different logic states on a feedback path and output. The example below shows simulation of pin feedback on I/O pins. In the example, OE is disabled for pin OUTA (an I/O pin). The feedback is then driven via OUTA.FB:

```
SETF/OE; disable OE  
SETFOUTA.FB; drive I/O feedback high  
SETFOE; driving I/O pin
```

The .FB extension does not need to be defined in the pin declarations or in any of the design sections. It is automatically understood by the simulator.

4. If a valid clock edge occurs during a register preload, the register states will be indeterminate.

Test Vector Notes

The following notes provide information that can help you create test vectors for use during programming verification.

1. Only the EP22V10 and EP22V10E support true register preloads during programming. Preloads in the simulation vector file are converted to JEDEC preload test vectors. If the programmer does not support preloads, you can not use preload simulation vectors when performing programming.
2. With the exception of the EP22V10 and EP22V10E, the test vectors in the JEDEC file for the preload simulation command (PRLDF) are output state values (SETFs), not true preload values. If you plan to use test vectors during programming for other devices, use the preload command only once at the beginning of the Simulation section to set registers to their power-up state. Use of the preload command elsewhere or with values other than the power-up value will cause warnings during the fitting process. Test vectors beginning with an illegal preload are not placed in the JEDEC file to eliminate the chance of programming errors. Note, however, that the programming test in this case can be much less thorough than expected.
3. During compilation, test vectors are automatically generated from the .HST file with the same base filename of the PLDasm file. If you do not include a simulation section in your current design file, but have a .HST file from a previous design that used the same filename, PLDasm will generate test vectors from the earlier design and include them in the JEDEC file for the new design. This will probably cause programming test failures. Deleting old .HST files when a design is complete will prevent this problem from occurring.

Estimating a Design

The estimator analyzes a design at a high level and provides a list of devices into which a given design may fit. Also, it lists devices rejected because they lack features or resources needed by the design.

- If the estimator says the design won't fit the device, then it won't.
- If the estimator says the design will fit the device, it may or may not fit, depending upon whether the selection/rejection of devices is based upon high-level criteria such as the number of I/Os, number of macrocells, number of equations, and the availability of device-dependent features such as SRAM, identity comparators, programmable output drivers and other requirements in a design. The estimator looks at the design on a high level (i.e., pins, p-terms, etc.). It does not look at aspects of the design that may be dependent upon certain architectural aspects of the device. The estimator will not warn of a no fit in these cases.

For example, no extensive SOP term tests or control term tests are performed. Thus, even though the estimator may report that the design is compatible with a specific device, the logic and interconnections may not fit. The task of determining a specific fit is left to the fitter.

Differences Between Estimating and Fitting

The estimator takes a high-level look at the overall utilization of device resources which are consumed by a design. The fitting process handles checking and resolving the more complex issues of signal routing, specific sizes of equations for control signals, the allocation of p-terms to macrocells, and other considerations.

Estimator Processing

The estimator performs the following checks to determine if the design will potentially fit into one of the devices in the database:

- Compares the number of inputs and outputs in the design to the number of active device pins.
- Compares the total number of output pins in the design to the number of device output pins.
- Compares the number of macrocells in the design to the number of device macrocells.
- Compares the number of p-terms in the design to the maximum number of device p-terms.
- Compares the size of p-terms in design to the maximum p-term size supported by the device.
- Checks if the types of outputs (register types or combinatorial) in the design are supported by the device.
- Checks if any special device features in the design are supported by the device, such as:
 - Open drain outputs
 - 3.3-volt/5-volt outputs
 - Input pull-ups
 - Identity compare
 - SRAM (length and width are also checked)
- Checks if the control signals used in the design are available in the device. This test checks whether a control is available as well as taking into account the number of

available control signal sources and control signal sharing. The control signals checked are:

- Synchronous or asynchronous clocks
- Register clear and preset
- Output enable

Interpreting Estimator Results

After estimation is completed, the devices which have passed the tests described earlier are listed as being potential candidates for a fit in the estimation report file (.EST). The list reports utilization percentages for the key resources consumed by the design and a “Total Device Utilization” percentage. A list of rejected devices is also generated which includes the first reason for failure.

There is some probability the design will not fit into the device(s) which pass the estimator’s tests, due to issues that can only be resolved by the fitter. The goal of the estimator is to provide a quick method of narrowing the choice of a target device for a design implementation. The estimator also gives the designer some idea of how much of a device’s resources will be utilized by a given design. This information, coupled with general guidelines on the ability of a device’s architecture to accommodate designs at various levels of resource utilization, can be a helpful indicator of which device to target for the final implementation.

It may be helpful to think of the estimator as a filter that eliminates devices lacking the necessary resources from consideration. This helps in making the choice of the most effective device from a device utilization standpoint.

Estimator Report File

The estimator report file is shown in Figure 5-8.

```

pldpick [ Vx.y ] SID [ x.y ]
DEVICE ESTIMATION REPORT FOR <FIF01>

*** DESIGN STATISTICS ***
Inputs ..... 14                # number of inputs in the design #
Outputs ..... 18                # number of outputs #
Active Pins (in+out) .... 32    # number of active pins #

Combinatorial.... 17            # combinatorial macrocells #
D flip-flops .... 19            # D flip-flops required #
T flip-flops .... 0             # T flip-flops required #
J/K (emulation) ... 0          # J/K flip-flops required#
Total Macrocells ..... 36      Buried Macrocells ... 18

Largest Product Term ... 10
SRAM Blocks ..... 1

DEVICE LISTED IN DESIGN: EPX780QC132
STATUS: OK

          ACTIVE          TOTAL          TOTAL
          PINS           MCELLS        PTERMS        DEVICE
          -----
EPX780QC132  32/104    36/80    22%          35%

# next two sections summarize how the resources of the devices checked would
# be consumed by the design                                     #

*** POTENTIAL DEVICES THAT MAY FIT FIF01 ***

          ACTIVE          TOTAL          TOTAL
          PINS           MCELLS        PTERMS        DEVICE
          -----
EPX740LC68   32/50     36/40    45%          73%
EPX780LC84   32/62     36/80    22%          42%
EPX780QC132  32/104    36/80    22%          35%

*** DEVICES REJECTED FOR FIF01 ***
EP220      Not enough input+output pins for this design
EP224      Not enough input+output pins for this design
EP22V10    Not enough input+output pins for this design
EP610      Not enough input+output pins for this design
EP910      Not enough macrocells for this design
EP312      Not enough input+output pins for this design
EP324      Not enough macrocells for this design
EPX740LC44 Not enough input+output pins for this design

```

Figure 5-8 Example of Estimator Report File

